MATS UNIVERSITY

NAAC GRADE A+ ACCREDITED UNIVERSITY

MATS CENTRE FOR DISTANCE & ONLINE EDUCATION

Web Technology

Bachelor of Computer Applications (BCA)
Semester - 4

SELF LEARNING MATERIAL

MATS UNIVERSITY
www.matsuniversity.ac.in

NAAC GRADE A+ ACCREDITED UNIVERSITY

Bachelor of Computer Applications

ODL BCA DSC- 401 T

Web Technology

COURSE DEVELOPMENT EXPERT COMMITTEE

Prof. (Dr.) K. P. Yadav, Vice Chancellor, MATS University, Raipur, Chhattisgarh

Prof. (Dr.) Omprakash Chandrakar, Professor and Head, School of Information Technology, MATSUniversity, Raipur, Chhattisgarh

Prof. (Dr.) Sanjay Kumar, Professor and Dean, Pt. Ravishankar Shukla University, Raipur, Chhattisgarh

Prof. (Dr.) Jatinderkumar R. Saini, Professor and Director, Symbiosis Institute of Computer Studies and Research, Pune

Dr. Ronak Panchal, Senior Data Scientist, Cognizant, Mumbai

Mr. Saurabh Chandrakar, Senior Software Engineer, Oracle Corporation, Hyderabad

COURSECOORDINATOR

Prof. (Dr.) Bhavna Narain, Professor, School of Information Technology, MATS University, Raipur, Chhattisgarh

COURSE PREPARATION

Prof. (Dr.) Bhavna Narain, Professor and Mr. Sanjay Behara, Assistant Professor, School of Information Technology, MATS University, Raipur, Chhattisgarh

March, 2025

ISBN: 978-93-49916-66-1

Printed & Published on behalf of MATS University, Village-Gullu, Aarang, Raipur by Mr. MeghanadhuduKatabathuni, Facilities & Operations, MATS University, Raipur (C.G.)

Printedat: TheDigitalPress, KrishnaComplex, Raipur-492001(Chhattisgarh)

Acknowledgement

The material (pictures and passages) we have used is purely for educational purposes. Every effort has been made to trace the copyright holders of material reproduced in this book. Should any infringement have occurred, the publishers and editors apologize and will be pleased to make the necessary corrections in future editions of this book.

COURSE INTRODUCTION

By this course provides an in-depth understanding of Web Technologies, focusing on the core components required to design, build, and manage dynamic and interactive websites. Covering foundational topics such as HTML, CSS, JavaScript, PHP, and version control systems, the course empowers learners to develop full-stack web applications using modern development practices. Through hands-on learning, students will gain proficiency in front-end and back-end technologies, understand integration through APIs, and apply version control for collaborative development.

Module 1: Introduction to HTML

This module introduces the foundational structure and syntax of HTML (HyperText Markup Language), the standard markup language for creating web pages. Learners will gain skills in using semantic tags, text formatting, tables, lists, file paths, iframes, and form elements to structure and present web content effectively. The module emphasizes building accessible and well-organized web pages.

Module 2: CSS – Cascading Style Sheets

CSS is integral to web presentation and design. This module explores the purpose, types, and application of CSS, including selectors, specificity, and inheritance rules. Learners will understand how to control layout and design using positioning techniques, backgrounds, borders, and overflow properties, enabling them to style and organize web content responsively and efficiently.

Module 3: JavaScript

This module focuses on the client-side scripting capabilities of JavaScript, enabling learners to enhance interactivity and dynamic behavior on web pages. Topics include variables, control structures, functions, DOM manipulation, and event handling. Learners will also be introduced to ES6 features such as arrow functions, promises, and block-scoped variables (let/const) for writing modern JavaScript code.

Module 4: PHP

Serving as the server-side scripting component, this module introduces PHP syntax, variables, and data types, along with form handling, session management, and database interaction using MySQL. Emphasis is placed on building dynamic, data-driven web applications. Additional topics include file handling, error management, basic security measures (e.g., preventing SQL injection), and an overview of popular PHP frameworks and performance optimization techniques.

Module 5: API, Git, and GitHub

Modern web development requires effective collaboration and integration. This module introduces the fundamentals of APIs, particularly RESTful APIs, and how to interact with them using Fetch API and Axios. It also provides a comprehensive introduction to Git version control, including branching, merging, pull requests, and conflict resolution, along with practical usage of GitHub for collaborative project management and GitHub Actions for automation and CI/CD workflows.By the end of this course, learners will have acquired the essential skills to design and develop fully functional, interactive, and collaborative web applications. They will be equipped to work confidently with both client-side and server-side technologies, integrate APIs, and manage development workflows using modern tools, preparing them for industry roles in web development and software engineering.

MODULE 1
INTRODUCTION TO HTML

LEARNING OUTCOMES

By the end of this module, learners will be able to:

- Understand the structure, elements, and syntax of HTML.
- Utilize text formatting and semantic tags to enhance web content.
- Implement IFRAMEs and manage file paths in HTML.
- Create and customize tables and lists for structured data representation.
- Design HTML forms with various input types, attributes, and validation techniques.

Unit 1: Foundational Concepts of HTML

1.1 Introduction to HTML – Structure, Elements, and Syntax

HTML Core: The World Wide Web would not exist without HyperText Markup Language (HTML), the basic building blocks used to create web pages. HTML, which is an abbreviation for Hyper Text Markup Language, was invented in 1993 by Tim Berners-Lee and has gone through several versions with HTML5 being the latest. HTML is a markup language, not a programming language, which means it is designed to annotate text so that a machine can process it and display the result. HTML works fundamentally through a system of elements indicated by these tags. These tags, which are surrounded by angle brackets, tell browsers how to interpret content. The opening tag, the content and the closing tag make up most of the HTML elements, it creates a nested tree-like structures that creates relationships between different parts of a document. This hierarchy is crucial to how browsers know how to parse and display web pages. There is a general template you will follow in constructing an HTML document. The doctype declaration The doctype declaration is the very first thing that appears in every HTML document and it specifies what version of HTML you are using. The declaration specifies HTML5, the latest specification. This triggers the beginnings of the document with the root element, followed by two major block elements, (the metadata and document-wide information not shown on the page) and (everything that users can see). Even though does not render any content, it has several significant roles. It contains important metadata that affects how a browser and search engine treat the page. The page title is defined by the element that gets displayed on the browser tabs and in the search results. Meta tags include character encoding, viewport settings for responsive design, and description for search engine optimization. The head section usually contains links to external resources like stylesheets, scripts, and favicons. Every single piece of content in HTML can be categorized as blocks, inline, or even replaced and there are a few that are self-contained variables that represent their data within a document. Heading elements (through), paragraphs, and divisions are block-level elements, which produce separate sections of content, while inline elements (, , and for example) change content without

producing new line breaks inside those blocks. To modify the behavior or appearance of HTML elements, you use HTML attributes special HTML words. Attributes are declared within the opening tag and can contain name-value pair (name="value").

Common properties are id(Unique Id to the component), class(for styling and JS selection), style (Inline CSS), and src(source of location) Some, like required for form elements, are boolean and require no value. The above example shows how an anchor () element has three attributes href (the url destination), target="_blank" (to show the link in a new tab), and title (the tooltip that appears when user hovers over the link). HTML comments, denoted by , let developers insert notes into the code that the browsers ignore. Comments in a code serve many purposes, such as to keep records about what the code is doing, to comment out a part of the code to see if it works without that part while an application is being developed and debugged. Proper nesting of elements is important for valid HTML. Elements are closed in the reverse order in which they are opened; a clean parent-child relationship is maintained. It can cause unpredictable rendering as well as accessibility problems, if nested improperly. Self-closing or void elements are a special kind of HTML elements that are never going to have content and don't need a closing tag. Some common examples would be image (), line break (), horizontal rule (), and input (). These elements can be written in simplified form without the trailing slash in HTML5, although it is still valid. The Document Object Model (DOM) is a standard in computing programming interface for HTML, which describes the structure of the page as a tree-like structure of objects that can receive commands via scripts. HTML's connection to the DOM is key to developing dynamic web content since JavaScript manipulates the DOM to change the document based on user actions. It is worth noting that HTML mainly describes structure, but as per the separation of concerns principle, it is more of a structure building block that works together with CSS (Cascading Style Sheets) for présentation and JavaScript for behaviour. This allows developers to change one thing without necessarily impacting the others, which increases maintainability. Valid HTML follows guidelines created by the World Wide Web Consortium (W3C). They point out syntax problems, bad nesting, etc., which may prevent browsers from

rendering properly. Modern browsers can usually handle broken HTML, but following the standards makes it safer and more accessible. Accessibility is one of the core aspects of HTML development. Semantic markup, correct heading structure, alt attribute for images, and ARIA (Accessible Rich Internet Applications) provide accessibility to users with disabilities (eg screen readers and assistive technologies). HTML will keep on evolving through new elements and attributes as web standards evolve. This evolution is mirrored in the transition from XHTML's rigid syntax requirements to HTML5's more lenient approach, which gives precedence of backward compatibility over other standards with modern web requirements. These are the basic concepts that will help you build a strong foundation for developing efficient, accessible, standards-compliant web documents.

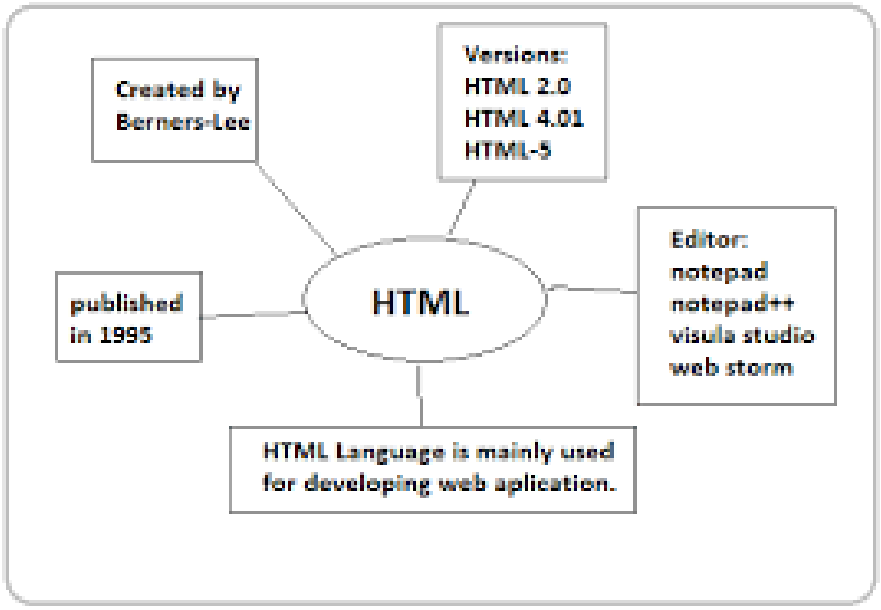

Figure 1.1 foundational Concepts Of HTML

(Source: https://www.fuundational.com)

Unit 2: Text Formatting, Semantic Markup, and Accessibility

1.2 Text Formatting and Semantic Tags in HTML

styles. meaning of the content they enclose. Realizing that semantic accessibility in HTML is a huge advancement in web development practices and not just as visual representation of where we write all the styles and elements, HTML gives us a bit of context about our data and allows us to prefix data with the structure which defines its meaning. There are two major categories of these: presentational elements that relate more to style, and semantic elements that indicate the Unlike traditional text formatting such as txt users navigate the content. level represents the importance of a heading, with , the most important, and , the least. Correct heading structure does more than help you visually perceive content hierarchy through default browser styling (larger font sizes for higher order headings), it also builds a semantic outline that assistive technologies can leverage to help the headings which defines the structure of the document as well as the hierarchy of the content. The heading elements, through , create a logical outline of the document, where each The next step to text formatting in HTML is by using or ideas. with nice whitespace. Paragraphs by default insert breaks before and after their content, visually separating distinct thoughts The paragraph element () makes up the most basic building block for generic text content, any collection of related text implies emphasized text that might be stressed if read aloud. in italics. These elements have semantic meaning beyond their visual display— suggests content of strong importance, urgency, seriousness, or formatting elements. The strong HTML element meaning strong importance usually in bold and the em HTML element meaning emphasized text usually To apply special treatment to specific parts of the text inside these blocks, HTML has several inline text important information and emphasized text. This paragraph includes is sample output from a computer program (all of these are technical content). side comments or small print, and is usually rendered in a smaller font size. is used for computer code, is keyboard input, and reference certain content. The element is for formatting needs. The element highlights text similar to a highlighter, and can be used to HTML also has elements for

some of the other common text (usually strikethrough for deletions and underline for insertions).

The <mark>highlighted section</mark> needs special attention.

<small>Terms and conditions apply.</small>

Use the <code>console.log()</code> function to debug.

Press <kbd>Ctrl+S</kbd> to save your document.

The program returned: <samp>Hello, World!</samp>

() elements lower or raise the text above or below the standard line, which is important in mathematical equations, chemical equations, and footnotes. The and elements represent deleted and inserted text respectively; they are often used to show document revisions along with appropriate visual clues Subscript () and superscript area formula is A = πr2. Water is H2O and. The meeting is on Tuesday Wednesday. The assistive technologies on how the content is organized and its purpose. contained content beyond just presentation. Newer HTML elements help developers make more structured, accessible documents while providing clearer signals to browsers, search engines, and After these primitive formatting elements, HTML5 added a very rich set of semantic elements that convey meaning about its complete, standalone component. blog. This tag indicates that the contents within it are a The element represents a self-contained composition where people could be interested, by itself or as a part of other compositions such as non-consecutive entries in a magazine, newspaper, website or

Article Title

Article content...

cover a different facet of the main theme. independent pieces of content, sections are rather parts of a larger whole. For example, you might have several sections on a single webpage that typically with a heading. Sections are like in that they can contain content, but instead of being The element is a thematic grouping of content,

Features

They can also be used inside introductory and concluding content for their nearest ancestor sectioning content or sectioning root element. Some elements are repeated across pages and are usually found at the top of a document (site branding, navigation, and search), and at Header and footer elements (and) represent of the product features...

description articles or sections as section-specific intro or outro content. the bottom of the document (copyright information, related links, and contact details).

```
<header>
  <h1>Website Title</h1>
  <nav>
    <ul>
      <li><a href="/">Home</a></li>
      <li><a href="/about">About</a></li>
    </ul>
  </nav>
</header>
<footer>
  <p>&copy; 2025 Example Corp. All rights reserved.</p>
</footer>
```

The navigation element (<nav>) specifically identifies major navigation blocks, helping browsers and assistive technologies recognize and potentially provide special access to these critical page components. This element typically contains lists of links to other pages or sections within the current page.

For complementary but non-essential content, the <aside> element indicates material related to the main content but which could be considered separate. Common uses include sidebars, pull quotes, advertising, and related article links.

```
<aside>
  <h3>Related Articles</h3>
  <ul>
    <li><a href="#">Similar Topic One</a></li>
    <li><a href="#">Similar Topic Two</a></li>
  </ul>
</aside>
```

The <figure> and <figcaption> elements work together to associate illustrations, diagrams, photos, or code listings with their captions, maintaining this relationship in a semantically meaningful way even if the position changes during layout.

```
<figure>
  <img src="diagram.png" alt="System architecture diagram">
```

<figcaption>Figure 1: Overview of system components and their interactions.</figcaption>
</figure>
For more specific text semantics, HTML provides elements like <address> for contact information, <time> for dates and times, and <cite> for references to creative works. These elements allow precise indication of content purpose beyond mere formatting.

```
<address>
  Written by <a href="mailto:john@example.com">John Doe</a>.<br>
  Visit us at: Example Corp<br>
  123 Main St, Anytown USA
</address>
<p>The concert starts at <time datetime="2025-04-07T20:00">8:00pm on April 7</time>.</p>
<p>As <cite>The HTML Specification</cite> states, the cite element represents the title of a work.</p>
```

Lists represent another fundamental aspect of text organization in HTML, with three main types available. Unordered lists (<ul>) present items with bullet points, ordered lists (<ol>) use sequential numbering, and description lists (<dl>) pair terms (<dt>) with their descriptions (<dd>). Lists can be nested to create hierarchical structures, and CSS can modify their presentation while maintaining semantic integrity.

```
<ul>
  <li>Unordered item one</li>
  <li>Unordered item two</li>
</ul>
<ol>
  <li>First step</li>
  <li>Second step</li>
</ol>
<dl>
  <dt>HTML</dt>
  <dd>HyperText Markup Language, the standard language for creating web pages.</dd>
  <dt>CSS</dt>
```

```
    <dd>Cascading Style Sheets, used for styling HTML
documents.</dd>
</dl>
```

For tabular data, the <table> element and its associated elements (<thead>, <tbody>, <tfoot>, <tr>, <th>, and <td>) provide structured organization. When used appropriately for genuinely tabular information rather than layout purposes, tables effectively present relationships between data points.

```
<table>
  <thead>
    <tr>
      <th>Name</th>
      <th>Age</th>
      <th>Location</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>John</td>
      <td>32</td>
      <td>New York</td>
    </tr>
    <tr>
      <td>Sarah</td>
      <td>28</td>
      <td>London</td>
    </tr>
  </tbody>
</table>
```

Blockquotes (<blockquote>) and inline quotes (<q>) format and semantically identify quoted content, with the optional cite attribute providing a URL reference to the source. These elements help distinguish quoted material from original content while maintaining proper citation relationships.

```
<blockquote cite="https://example.com/source">
    <p>The only limit to our realization of tomorrow will be our doubts
of today.</p>
</blockquote>
```

As Einstein said, <q cite="https://example.com/einstein">imagination is more important than knowledge</q>.

for styling. that affected appearance, but no link to the content meaning. Modern best practices eschew these in favor of semantic equivalents coupled with CSS an important evolution in web development practices. Early flavor of HTML had purely presentational elements , , (bold) elements And semantic HTML versus presentational HTML presents CSS font-style for display italics. for important content or to use CSS font-weight on elements where the bold does not indicate importance. Likewise, and instead of for italic text, should be used for emphasized content, or Furthermore, instead of using for bold text, developers should apply strong

Important warning

Warning:

but not semantically important Visually bold

Having party-based and index-based communication enables you to make the content Developers can leverage the text formatting features of HTML and use semantic elements to create contributes to responsive design as it improves the readability of content, allowing it to scale well on various devices and screen sizes. for users with disabilities. Semantic HTML within semantic elements, particularly headings, to enhance the relevance of search results. Screen readers, for example, rely on semantic information to enable enhanced navigation and content understanding to theoretical purity. Search engines place more importance on content The advantages of semantic HTML are not just an appeal accessibility technique in the Web Content Accessibility Guidelines (WCAG) [WCAG 2008]. type of element (such as headings, lists or navigation) which helps the users to understand the structure of content and navigate accordingly. Semantic markup is recommended as a core beneficial to accessibility. Screen readers read out the Proper semantic markup is particularly s with class names. code. Thus, for displaying actual code reviews, elements like , , and convey their roles instantly and aid in comprehension of a document hierarchy compared to repeating generic For super devs, semantic HTML helps improve the maintainability and readability of the text formatting and semantics is

still the way to create accessible, maintainable and usable web content. the DOM. How to use appropriate markup. Make no mistake, priest architect, even the most modern JavaScript framework with their components based architecture still depends on valid HTML semantics which they use as a basis to generate dynamic structures in This, along with the continual evolution of web development, has only increased the emphasis on semantic ultimately serves user experience and development experiences alike. visual styling, separating content structure from presentation. This creates cleaner and more accessible code that select HTML elements based primarily on its semantic meaning rather than default presentation. CSS should take care of this However, in practice, developers should available and valuable on all devices, browsers or user needs, which is what web aspiration as a universal information platform. documents that convey meaning both visually and contextually.

1.3 IFRAME and File Path Handling in HTML

to handle filepath correctly a basic life skill in a web developer! iframe Element In this section we learn all about the wonders of iframes as well as how The Most Powerful Element in HTML: the

Understanding IFrames

simple: to another webpage from the current page. Iframe syntax is referred to as iframe, is used to embed another document within the current HTML document. This enables developers to place one HTML document within another, kind of like a window An inline frame, commonly Sandbox Attribute Is Important in Preventing These Risks: attacks. The of iframes The iframe may also cause security vulnerabilities if not implemented correctly. Embedding harmful contents can lead to cross-site scripting (XSS) attribute defines the URL of the document to embed, and other attributes are used for specifying different properties and behaviors of the iframe. Below are the most important iframe attributes you may use for customization:

- **width and height:** set respective dimensions of your iframe
- **name:** provides a name for your iframe to be targeted in links or scripts
- **sandbox**: extra restrictions on operations within the iframe for extra security
- **allow:** Feature that is accessible within the iframe (like the camera, microphone)

- **loading:** Determines how the iframe is loaded (eager or lazy)
- **Security aspects The srcdoc**: An HTML code to display other than by loading a web page Therefore, to embed a YouTube video as a responsive iframe with specific dimensions and lazy loading

The sandbox attribute restricts various capabilities of the embedded content, with options to selectively enable certain features. Common values include:

- **allow-scripts**: Permits JavaScript execution
- **allow-same-origin**: Allows the content to be treated as same-origin
- **allow-forms**: Enables form submission
- **allow-popups**: Allows popup windows
- **allow-top-navigation**: Permits navigation of the top-level browsing context

Additionally, the Content Security Policy (CSP) header can be used to control which sources are allowed to be embedded in iframes:

```
<meta http-equiv="Content-Security-Policy" content="frame-src 'self' https://trusted-site.com;">
```

Responsive IFrames

Making iframes responsive is essential for modern web design. A common technique involves wrapping the iframe in a container with appropriate CSS:

```
<div class="iframe-container">
  <iframe src="embedded-content.html"></iframe>
</div>
```

With corresponding CSS:

```
.iframe-container {
  position: relative;
  width: 100%;
  padding-bottom: 56.25%; /* 16:9 aspect ratio */
  height: 0;
  overflow: hidden;
}
.iframe-container iframe {
  position: absolute;
  top: 0;
  left: 0;
```

```
  width: 100%;
  height: 100%;
  border: 0;
}
```

This approach maintains a specific aspect ratio while allowing the iframe to scale with its container.

File Path Handling in HTML

Proper file path handling is essential for organizing web projects and ensuring resources are correctly linked. HTML uses file paths to reference various resources such as images, stylesheets, scripts, and other HTML documents.

Types of File Paths

There are several types of file paths used in HTML:

1. Absolute Paths

Absolute paths provide the complete URL to a resource, including the protocol and domain:

```
<img src="https://example.com/images/logo.png" alt="Logo">
<link rel="stylesheet" href="https://example.com/css/styles.css">
```

These paths work regardless of the current document's location but require the resource to be accessible at the specified URL.

2. Relative Paths

Relative paths specify the location of a resource relative to the current document:

```
<!-- Same directory -->
<img src="logo.png" alt="Logo">
<!-- Subdirectory -->
<img src="images/logo.png" alt="Logo">
<!-- Parent directory -->
<img src="../images/logo.png" alt="Logo">
```

Relative paths are more portable since they don't depend on specific domains, making them ideal for local development and projects that might be moved between servers.

3. Root-Relative Paths

Root-relative paths start with a forward slash and are relative to the domain root:

```
<img src="/images/logo.png" alt="Logo">
<link rel="stylesheet" href="/css/styles.css">
```

These paths work consistently across all pages of a website, regardless of their directory depth.

Best Practices for File Path Organization

Organizing files effectively is crucial for maintainable web projects:

1. Consistent Directory Structure: Maintain a logical hierarchy with separate directories for different types of resources:
2. /project
3. /css
4. /js
5. /images
6. /pages
7. index.html
8. Descriptive File Names: Use clear, descriptive file names that indicate content and purpose.
9. Case Sensitivity: Be aware that some servers (particularly Unix-based) treat file names as case-sensitive.
10. Path Types for Different Scenarios:
    - Use relative paths for resources within the same project
    - Use absolute paths for external resources
    - Use root-relative paths for resources that should be accessible from any page on your site
11. URL Encoding: Ensure special characters in file paths are properly URL-encoded:
12. <img src="images/my%20image.jpg" alt="My Image">

Common Path-Related Issues and Solutions

Several common issues arise with file paths:

1. **Broken Links**: Often caused by incorrect relative paths. Use browser developer tools to identify and fix broken resource links.
2. **Path Traversal Vulnerabilities**: Sanitize user input used in file paths to prevent attackers from accessing unauthorized files.
3. **Cross-Origin Resource Sharing (CORS)**: When loading resources from different domains, be aware of CORS restrictions and ensure proper server headers.
4. **Moving Projects Between Environments**: Use relative paths within projects to facilitate movement between development, staging, and production environments.

Using File Paths with IFrames

File paths are particularly important when working with iframes, as they determine which document gets embedded:

```
<!-- Embedding a document from the same directory -->
<iframe src="related-content.html"></iframe>
<!-- Embedding from a subdirectory -->
<iframe src="pages/related-content.html"></iframe>
<!-- Embedding from parent directory -->
<iframe src="../other-section/related-content.html"></iframe>
<!-- Embedding from an external site -->
<iframe src="https://example.com/embed-page.html"></iframe>
```

When embedding content from the same project, relative paths provide flexibility. For external content, absolute paths are necessary.

Advanced IFrame Techniques

Beyond basic implementation, iframes offer advanced capabilities that enhance their utility in modern web development.

Communication Between IFrames and Parent Documents

The postMessage API enables secure communication between an iframe and its parent document:

```
// In the parent document
const iframe = document.querySelector('iframe');
iframe.contentWindow.postMessage('Hello from parent!', 'https://embedded-site.com');
// In the iframe
window.addEventListener('message', function(event) {
  if (event.origin === 'https://parent-site.com') {
    console.log('Message from parent:', event.data);
  }
});
```

This communication method is crucial for creating interactive embedded applications while maintaining security boundaries.

Lazy Loading IFrames

To improve page performance, iframes can be lazy-loaded to defer loading until they're needed:

```
<iframe src="heavy-content.html" loading="lazy"></iframe>
```

For browsers that don't support the loading attribute, JavaScript alternatives can be implemented:

```
const observer = new IntersectionObserver((entries) => {
```

```
  entries.forEach(entry => {
    if (entry.isIntersecting) {
      entry.target.src = entry.target.dataset.src;
      observer.unobserve(entry.target);
    }
  });
});
document.querySelectorAll('iframe[data-src]').forEach(iframe => {
  observer.observe(iframe);
});
```

Seamless Integration with CSS

To make iframes appear more integrated with the parent document, CSS techniques can be employed:

```
iframe {
  border: none;
  background-color: transparent;
  overflow: hidden;
}
```

This removes the typical iframe border and creates a more seamless experience.

File Path Best Practices for Project Scalability

As web projects grow, maintaining proper file paths becomes increasingly important for scalability and maintainability.

Using Base URLs

The <base> element can simplify file path handling by specifying a base URL for all relative URLs in a document:

```
<head>
  <base href="https://example.com/projects/current-project/">
  <!-- All relative paths will be resolved against the base URL -->
  <link rel="stylesheet" href="css/styles.css">
  <!-- Equivalent to https://example.com/projects/current-project/css/styles.css -->
</head>
```

This approach is particularly useful for sites with complex directory structures or those that might be moved to different locations.

Dynamic Path Generation

For large-scale applications, dynamically generating paths through server-side languages or JavaScript can provide flexibility:

```
<!-- PHP example -->
<img src="<?php echo $basePath; ?>/images/logo.png" alt="Logo">
// JavaScript example
document.querySelector('iframe').src = `${baseUrl}/embedded-content.html`;
```

This approach allows path configurations to be centralized and easily updated.

In conclusion, mastering iframes and file path handling is essential for creating well-structured, secure, and maintainable web projects. These elements provide the foundation for effectively organizing and embedding content, enabling developers to create rich, interactive web experiences while maintaining proper resource management.

1.4 Tables and Lists – Creation and Customization

Tables and lists are fundamental HTML elements that organize and present information in structured formats. From simple data presentation to complex layouts, understanding how to create and customize these elements is essential for effective web development.

HTML Tables: Structure and Semantics

Tables in HTML are designed to present tabular data—information organized in rows and columns. The fundamental structure of an HTML table consists of several key elements:

```
<table>
 <caption>Monthly Sales Data</caption>
 <thead>
  <tr>
   <th>Month</th>
   <th>Sales</th>
   <th>Growth</th>
  </tr>
 </thead>
 <tbody>
  <tr>
   <td>January</td>
   <td>$10,000</td>
   <td>5%</td>
  </tr>
  <tr>
   <td>February</td>
```

```
    <td>$12,500</td>
    <td>25%</td>
   </tr>
  </tbody>
  <tfoot>
   <tr>
    <td>Total</td>
    <td>$22,500</td>
    <td>-</td>
   </tr>
  </tfoot>
</table>
```

Semantic Table Elements

Modern HTML tables include several semantic elements that enhance accessibility and structure:

- **<caption>**: Provides a title or explanation for the table
- **<thead>**: Contains header rows (<tr>) with column headers
- **<tbody>**: Contains the main data rows
- **<tfoot>**: Contains footer rows, typically for summaries or totals
- **<th>**: Defines header cells, which should use the scope attribute to specify whether they're for rows or columns
- **<td>**: Defines standard data cells

These semantic elements not only structure the table visually but also provide important information to screen readers and other assistive technologies.

Table Attributes for Structure

Several attributes can modify the structure of tables:

- **colspan**: Allows a cell to span multiple columns
- <td colspan="2">Combined Cell</td>
- **rowspan**: Allows a cell to span multiple rows
- <td rowspan="3">Spans Three Rows</td>
- **headers**: Associates data cells with their corresponding headers for accessibility
- <th id="name">Name</th>
- <td headers="name">John Doe</td>

Table Accessibility Considerations

Creating accessible tables is crucial for users with disabilities:

1. **Use proper header cells**: Always use <th> elements for headers with appropriate scope attributes:
2. <th scope="col">Product</th>
3. <th scope="row">Q1 Sales</th>
4. **Include captions**: Provide context with the <caption> element
5. **Use the <table> element only for tabular data**: Tables should not be used for layout purposes
6. **Provide summary information**: For complex tables, use ARIA attributes:
7. <table aria-describedby="table-summary">
8. <!-- table content -->
9. </table>
10. This table shows quarterly sales data across regions.

Styling Tables with CSS

Basic HTML tables are functional but often lack visual appeal. CSS transforms tables into attractive, readable components.

Basic Table Styling

Simple CSS properties can dramatically improve table appearance:

```
table {
  border-collapse: collapse;
  width: 100%;
  margin-bottom: 1em;
  font-family: Arial, sans-serif;
}
th, td {
  padding: 0.75em;
  text-align: left;
  border-bottom: 1px solid #ddd;
}
th {
  background-color: #f2f2f2;
  font-weight: bold;
}
tr:hover {
  background-color: #f5f5f5;
```

```
}
```

Zebra Striping

Alternating row colors improves readability:

```
tr:nth-child(even) {
  background-color: #f2f2f2;
}
```

Responsive Tables

Tables can be challenging on small screens. Several techniques address this:

1. **Horizontal scrolling**:

```
.table-container {
  overflow-x: auto;
}
Collapsing rows on small screens:
@media screen and (max-width: 600px) {
  table, thead, tbody, th, td, tr {
    display: block;
  }

  thead tr {
    position: absolute;
    top: -9999px;
    left: -9999px;
  }

  td {
    position: relative;
    padding-left: 50%;
    border: none;
    border-bottom: 1px solid #eee;
  }

  td:before {
    position: absolute;
    left: 6px;
    width: 45%;
    padding-right: 10px;
    white-space: nowrap;
```

```
    content: attr(data-label);
    font-weight: bold;
  }
}
```

With corresponding HTML that includes data attributes:

```
<td data-label="Month">January</td>
```

```
Using CSS Grid for responsive layouts:
@media screen and (max-width: 600px) {
  table {
    display: grid;
  }

  tr {
    display: grid;
    grid-template-columns: 1fr 1fr;
    margin-bottom: 1em;
    border: 1px solid #ddd;
  }

  th, td {
    padding: 0.5em;
  }
}
```

Advanced Table Styling Techniques

Beyond basic styling, tables can benefit from advanced CSS:

1. **Fixed headers with scrollable body**:

```
.table-container {
  height: 300px;
  overflow-y: auto;
}
thead {
  position: sticky;
  top: 0;
  background-color: #fff;
  z-index: 1;
}
Column width control:
table {
```

```
14.  table-layout: fixed;
15. }
16. th:nth-child(1) {
17.  width: 20%;
18. }
19. th:nth-child(2) {
20.  width: 50%;
21. }
22. Cell text handling:
23. td {
24.  white-space: nowrap;
25.  overflow: hidden;
26.  text-overflow: ellipsis;
27. }
```

HTML Lists: Types and Structures

HTML offers three primary types of lists, each with distinct semantic meaning and visual presentation.

Unordered Lists

Unordered lists (<ul>) present items in no particular order, typically with bullet points:

```
<ul>
  <li>Apples</li>
  <li>Oranges</li>
  <li>Bananas</li>
</ul>
```

Ordered Lists

Ordered lists (<ol>) present items in a specific sequence, using numbers, letters, or Roman numerals:

```
<ol>
  <li>Preheat oven to 350°F</li>
  <li>Mix ingredients in a bowl</li>
  <li>Bake for 25 minutes</li>
</ol>
```

Description Lists

Description lists (<dl>) associate terms (<dt>) with their descriptions (<dd>):

```
<dl>
  <dt>HTML</dt>
```

```
  <dd>HyperText Markup Language, the standard language for web pages</dd>

  <dt>CSS</dt>
  <dd>Cascading Style Sheets, used for styling web pages</dd>
</dl>
```

Nested Lists

Lists can be nested inside one another to create hierarchical structures:

```
<ul>
  <li>Fruits
    <ul>
      <li>Apples</li>
      <li>Oranges</li>
    </ul>
  </li>
  <li>Vegetables
    <ul>
      <li>Carrots</li>
      <li>Broccoli</li>
    </ul>
  </li>
</ul>
```

Customizing Lists with CSS

List presentation can be extensively customized using CSS.

Basic List Styling

Simple properties can transform the appearance of lists:

```
ul, ol {
  margin: 0 0 1em 0;
  padding-left: 2em;
}
li {
  margin-bottom: 0.5em;
  line-height: 1.4;
}
dl {
  margin: 0 0 1em 0;
}
dt {
```

```
  font-weight: bold;
  margin-bottom: 0.25em;
}
dd {
  margin: 0 0 1em 2em;
}
```

Custom List Markers

The list-style-type property controls the appearance of list markers:

```
ul {
  list-style-type: square; /* Options: disc, circle, square, none */
}
ol {
  list-style-type: lower-roman; /* Options: decimal, upper-alpha, lower-alpha, upper-roman, lower-roman */
}
```

Custom markers can be created using images or Unicode characters:

```
ul {
  list-style-image: url('bullet.png');
}
/* Or using ::marker */
li::marker {
  content: "➔ ";
  color: #007bff;
}
/* Or removing default markers and adding custom ones */
ul {
  list-style: none;
  padding-left: 0;
}
li::before {
  content: "★";
color: gold;
display: inline-block;
width: 1em;
margin-left: -1em;
}
```

Horizontal Lists

can become horizontal navigation menus: Lists

```
ul.horizontal {
list-style: none;
padding: 0;
margin: 0;
display: flex;
}
ul.horizontal li {
margin-right: 1em;
}
using inline-block */  Alternative
ul.horizontal-alt {
list-style: none;
padding: 0;
margin: 0;
}
ul.horizontal-alt li {
display: inline-block;
margin-right: 1em;
}
```

Interactive List Elements

add interaction to the list items: Here are several ways to

```
ul.interactive li {
padding: 0.5em 1em;
border-radius: 4px;
background-color.3s;  transition:
}
ul.interactive li:hover {
background-color: #f0f0f0;
cursor: pointer;
}
```

Multi-column Lists

readability for long lists: Multiple columns aid

```
ul.multi-column {
column-count: 3;
column-gap: 2em;
list-style-position: inside;
}
```

Prevent items from spanning multiple columns * / / *

```
ul.multi-column li {
break-inside: avoid;
}
```

JavaScript is necessary: for Tables and Lists The Original Functionality specialised use. Tables and list are not just for their basic functions, they can be moulded into a for interactive functionalities, but CSS does prepare tables for interactivityth.sortable

```
{
cursor: pointer;
position: relative;
}
th.sortable::after {
content: "↕";
position: absolute;
right: 8px;
color: #999;
}
th.sorted-asc::after {
content: "↑";
color: #333;
}
th.sorted-desc::after {
content: "↓";
color: #333;
}
tr.filtered {
display: none;
}
```

Lists as Navigation Menus

into a list structure: Navigation elements naturally fall

Home

Products

Category 1

Category 2

About

Contact

With supporting CSS:

```
.main-menu {
```

```
list-style: none;
padding: 0;
margin: 0;
display: flex;
background-color: #333;
}
.main-menu > li {
position: relative;
}
.main-menu a {
display: block;
padding: 1em 1.5em;
color: white;
text-decoration: none;
}
.main-menu a:hover {
background-color: #555;
}
.dropdown {
display: none;
position: absolute;
background-color: #444;
min-width: 200px;
0 8px 16px rgba(0,0,0,0.2);  box-shadow: box-shadow:
z-index: 1;
list-style: none;
padding: 0;
}
. has-dropdown:hover. dropdown {
display: block;
}
.dropdown a {
padding: 0.75em 1.5em;
}
```

Combining Tables and Lists

visualization: Here is how to combine tables and lists for more complex data

Department

Team Members
Projects
Marketing
John Smith (Lead)
Maria Garcia
David Chen
Website Redesign
Media Campaign）】 Do上社會媒介【（Social
Development
Sarah Johnson (Lead)
Michael Brown
Ana Rodriguez
API Integration
Mobile App
Bug Fixes

CSS Grid vs. Tables

layout: Tables — use only for tabular data! CSS Grid is an alternative
for

```
.grid-table {
display: grid;
repeat(3, 1 fr);  grid-template-columns:
gap: 1em;
margin-bottom: 1em;
}
.grid-table-header {
font-weight: bold;
background-color: #f2f2f2;
padding: 0.75em;
}
.grid-table-cell {
padding: 0.75em;
solid #ddd;}  border-bottom: 1px
}
```

With corresponding HTML:

```
Name
Title
Department
Doe  John
```

Manager
do the same in an MVC → How to
Developer
Engineering
uses table-shaped layouts but doesn't run into semantic problems that come with using a for non-table content. This

Best Practices and Performance Considerations

Creating efficient, accessible tables and lists requires attention to best practices.

Table Performance Optimization

Large tables can affect page performance:

1. **Lazy loading for large datasets**: Load only visible portions initially
2. **Virtual scrolling**: Render only visible rows and replace them as the user scrolls
3. **Pagination**: Split large datasets across multiple pages
4. **Minimizing DOM elements**: Avoid unnecessary nested elements within table cells
5. **CSS optimization**: Use efficient selectors and minimize repaints/reflows

```
<div class="table-pagination">
  <button id="prev-page">Previous</button>
  <span>Page <span id="current-page">1</span> of <span id="total-pages">5</span></span>
  <button id="next-page">Next</button>
</div>
```

List Performance Optimization

For large lists:

1. **Fragment caching**: For server-rendered lists with frequent updates
2. **Virtualization**: Similar to tables, render only visible items
3. **Limiting animation complexity**: Animations on list items can cause performance issues
4. **Debouncing scroll events**: When implementing custom scroll behavior

Cross-Browser Compatibility

Ensuring consistent appearance across browsers:

1. **CSS resets or normalizers**: Standardize default styling

2. **Vendor prefixes**: For newer CSS properties
3. **Feature detection**: Test for supported features before using them
4. **Polyfills**: For newer features in older browsers

```
/* Example of vendor prefixes for sticky positioning */
thead {
  position: -webkit-sticky;
  position: -moz-sticky;
  position: -ms-sticky;
  position: -o-sticky;
  position: sticky;
  top: 0;
}
```

Accessibility Beyond the Basics

Beyond fundamental accessibility considerations:

1. Custom keyboard navigation: For interactive tables or lists
2. // Example of arrow key navigation in a table
3. table.addEventListener('keydown', function(e) {
4. if (e.key === 'ArrowDown' || e.key === 'ArrowUp' ||
5. e.key === 'ArrowLeft' || e.key === 'ArrowRight') {
6. // Handle navigation between cells
7. }
8. });
9. ARIA live regions: For dynamic content updates
10.
11. Table data updated.
12.
13. High contrast modes: Test and optimize for Windows High Contrast Mode
14. @media (forced-colors: active) {
15. th, td {
16. border: 1px solid CanvasText;
17. }
18. }
19. Reduced motion preferences: Respect user preferences for animations
20. @media (prefers-reduced-motion: reduce) {
21. .animated-list li {

```
    transition: none;
  }
}
```

Emerging Trends and Future Directions

The evolution of tables and lists continues with emerging standards and techniques.

Modern CSS Features for Tables and Lists

1. Container queries: Style tables and lists based on their container's size rather than the viewport

```
@container (min-width: 500px) {
  .responsive-table {
    /* Styles for wider containers */
  }
}
Subgrid: Allow table cells to participate in a parent grid layout
.table-grid {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
}
.table-row {
  display: grid;
  grid-template-columns: subgrid;
  grid-column: 1 / -1;
}
:has() selector: Style parent elements based on child content
/* Style rows that contain empty cells */
tr:has(td:empty) {
  background-color: #ffeeee;
}
```

Data Visualization Evolution

Tables and lists are evolving into richer data visualization components:

1. **Interactive data tables**: With built-in sorting, filtering, and inline editing
2. **Hybrid list-chart components**: Combining textual data with visual representations
3. **Animated transitions**: Smooth state changes when data updates

4. **Contextual information**: Tooltips, popovers, and expandable rows for additional details

Accessibility Trends

The focus on inclusive design continues to shape tables and lists:

1. Standardized keyboard interaction patterns
2. Enhanced screen reader annotations
3. Voice navigation support
4. Internationalization considerations

In conclusion, tables and lists remain fundamental HTML elements with endless possibilities for customization and enhancement. By mastering these elements and applying modern CSS techniques, developers can create accessible, responsive, and visually appealing components that effectively organize and present information. Whether used for data presentation, navigation, or complex interactive interfaces, well-crafted tables and lists enhance usability and user experience across diverse devices and contexts.

1.5 HTML Forms – Input Types, Attributes, and Validation

HTML Forms are the crux of any interactive web application. Forms allow for structured ways for a user to submit information that can be processed by the web server or client-side scripts. In this article, we will analyze the properties of HTML forms, including input types, attributes, and application of validation techniques.

Form Basics

Finally, an HTML form is created with the element to hold the various input elements. A form has this elementary structure:

- Give an example of quality data you can use with your responses.

Now all you have to do is synchronize.

The action attribute specifies where to send the form data when the form is submitted, and the method attribute defines how to send the data (the two most common ways are "get" and "post").

Input Types

All the new input types introduced in HTML5. There's a unique interface for each input type that takes a certain kind of information:

Text-Based Inputs

- **text:** Regular one line text input
- **password:** Input of masked text for sensitive data
- **email:** Email address optimized input field with inline validation

- **searching:** Input box to enter search terms
- **tel:** Telephone numbers input field
- **url:** URL input element with automatic validation
- **textarea:** Multiline text input

Selection Inputs

- **checkbox:** Let users choose multiple options
- **radio:** Users can pick a single choice from a list
- **select:** Generates a list from which the user must select an option
- **datalist:** A predefined list for sharing options in an input field

Specialized Inputs

- **number:** Input for numbers with up/down buttons
- **range:** A slider control for choosing a value from a range
- **color:** a color picker interface
- **date:** Calendar picker
- **time:** Time selector
- **datetime-local:** Date and time picker in one
- **month:** Month and year picker
- **week:** Week and year picker
- **file:** File upload interface

Button Inputs

- **button:** Generic button for custom JavaScript actions
- **submit:** The button thatSubmit the form
- **reset:** A button that resets a form fields back to the default values
- **image**: Graphical submit button

Form Attributes

There are several attributes that HTML form elements support which controls their behavior and appearance:

Common Input Attributes

- **name:** Identifies the input field when data is submitted
- **id:** Identifier for the element (one per element; needed for labels and scripts)
- **value:** Default/current value of the input
- **placeholder:** Hint text shown when the input is empty
- **required:** The field is required to be filled out
- **disabled :** Makes input unusable and not submitted when submitting form

- **readonly:** Puts the input in a non-editable state, while still submitting it with a form
- **autofocus:** Focuses the input when the page is opened
- **autocomplete:** Controls browser autocomplete functionality
- **width:** Sets the width of text inputs
- **maxlength:** the maximum length of the input character
- **min and max:** Sets minimum and maximum values for range and number inputs
- **step**: Sets the increment/decrement step for number and range inputs.

Form-Specific Attributes

- **enctype:** Specifies how form data should be encoded (important for file uploads)
- **novalidate**: Disables browser validation for the form
- **target:** Determines where the form result is displayed
- **autocomplete:** Controls browser autocomplete functionality at the form level

Form Labels and Accessibility

Proper labeling of form elements is crucial for accessibility and usability:

```
<label for="username">Username:</label>
<input type="text" id="username" name="username">
```

The for attribute in the label links it to the input field with the matching id. This association allows users to click on the label to focus the input, which is particularly helpful for checkboxes and radio buttons.

Form Validation

HTML5 introduced built-in form validation features that allow developers to enforce data integrity before submission:

Validation Attributes

- **required:** Ensures that a field is not empty
- **pattern:** Applies a regular expression pattern for validation
- **minlength and maxlength:** Controls text length
- **min and max:** Restricts numerical values
- **step:** Enforces value increments
- **type**: Provides basic validation (e.g., email, url, number)

Example of Form Validation

```
<form>
   <label for="email">Email:</label>
   <input type="email" id="email" name="email" required>

   <label for="password">Password:</label>
   <input type="password" id="password" name="password"
        minlength="8" required
        pattern="(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{8,}"
        title="Must contain at least one number and one uppercase and lowercase letter, and at least 8 or more characters">

   <input type="submit" value="Submit">
</form>
```

Constraint Validation API

JavaScript can access the Constraint Validation API to perform custom validation:

```
const email = document.getElementById('email');
email.addEventListener('input', function() {
   if (email.validity.typeMismatch) {
      email.setCustomValidity('Please enter a valid email address');
   } else {
      email.setCustomValidity('');
   }
});
```

Form Organization and Structure

For more complex forms, grouping related elements improves usability and accessibility:

Fieldset and Legend

Details Personal

Name:

Email:

Shipping Address for

address Input type text name address id address

or clear the boards. You can change your board to too many input fields

City:

Submit

Advanced Form Features

to improve the user experience: HTML5 comes with several new form advanced features

Datalist Autocomplete Suggestions

from; INPUT List of values to pick

Output Element

displays the result of a calculation or user action: The element

=

100

Form Data Handling

JavaScript is commonly used in modern web applications to handle form submissions, as shown below, which prevents the default form submission behavior and processes the data asynchronously:

function(event) { document. querySelector('form'). addEventListener('submit',

event.preventDefault();

FormData(this); var formData = new

fetch('/submit-form', {

method: 'POST',

body: formData

})

. then(response => response. json())

. then(data => console. log(data))

. catch(error => console. error(error));

});

A Comprehensive Tutorial on Input Types and form inputs provides the flexibility needed to meet diverse user requirements. ValidationHTML forms have advanced a lot since the beginning of the platform. Forms in HTML are fundamental to user interaction and data collection on websites, and the ability to create a wide variety of Data Collection With HTML5 Forms.

1.6 HTML Layout – Head, ID, Class, and CSS Integration

Official documentation Learn about HTML Creating organized, accessible, and are used to create coherent web pages. layout elements The document head section CSS for HTML In this section we will discuss these building blocks and how they visually pleasing web pages relies on an understanding of the layout and formatting of HTML documents.

The HTML Document Structure

A well-structured HTML document follows a standard organization that separates document metadata from content:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <!-- Document metadata goes here -->
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document Title</title>
   <link rel="stylesheet" href="styles.css">
   <script src="script.js" defer></script>
</head>
<body>
   <!-- Document content goes here -->
   <header>
      <h1>Website Title</h1>
      <nav>
         <!-- Navigation links -->
      </nav>
   </header>

   <main>
      <section id="about">
         <!-- Section content -->
      </section>

      <article class="blog-post">
         <!-- Article content -->
      </article>
   </main>

   <footer>
      <!-- Footer content -->
   </footer>
</body>
</html>
```

The Head Section

The <head> section of an HTML document contains metadata that is not displayed on the page but provides crucial information about the document:

Essential Head Elements

1. **Document Title**: The <title> element defines the page title displayed in the browser tab.

<title>My Website - Home Page</title>

2. **Character Encoding**: The <meta charset> tag specifies the character encoding for the document.

<meta charset="UTF-8">

3. **Viewport Settings**: The viewport meta tag controls how the page is displayed on mobile devices.

<meta name="viewport" content="width=device-width, initial-scale=1.0">

4. **SEO Metadata**: Meta tags for search engine optimization provide information about the page content.

<meta name="description" content="A comprehensive guide to HTML layouts and CSS integration">
<meta name="keywords" content="HTML, CSS, web development, layout">
<meta name="author" content="John Doe">

5. **External Resource Links**: The <link> element connects the HTML document to external resources, most commonly CSS stylesheets.

<link rel="stylesheet" href="styles.css">
<link rel="icon" href="favicon.ico" type="image/x-icon">
<link rel="preconnect" href="https://fonts.googleapis.com">

6. **Scripts**: The <script> element includes JavaScript code or links to external JavaScript files.

<script src="script.js" defer></script>

7. **Style Definitions**: The <style> element contains internal CSS rules.

```
<style>
  body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
```

```
    }
</style>
```

HTML IDs and Classes

IDs and classes are attributes that allow developers to identify and target specific HTML elements for styling and scripting purposes:

ID Attribute

The id attribute provides a unique identifier for an HTML element. Each ID must be unique within the document:

```
<div id="header">
    <h1>Website Title</h1>
</div>
```

IDs are referenced in CSS with the hash symbol (#):

```
#header {
    background-color: #333;
    color: white;
    padding: 20px;
}
```

IDs are also used for:

- Creating anchor links within a page
- JavaScript manipulation via getElementById()
- Form label associations
- WAI-ARIA relationships

Class Attribute

The class attribute assigns one or more classification names to an element. Multiple elements can share the same class, and a single element can have multiple classes:

```
<article class="post featured">
    <h2>Article Title</h2>
    <p class="summary">Article summary text...</p>
</article>
```

Classes are referenced in CSS with a period (.):

```
.post {
    margin-bottom: 20px;
    border: 1px solid #ddd;
}
.featured {
    background-color: #f8f8f8;
    border-left: 5px solid #007bff;
```

```
}
.summary {
   font-weight: bold;
   font-style: italic;
}
```

Classes provide a powerful way to:

- Apply consistent styling across multiple elements
- Group elements for JavaScript manipulation
- Implement design systems with reusable components
- Create state-based styling (e.g., .active, .disabled)

CSS Integration

CSS (Cascading Style Sheets) can be integrated with HTML in three ways:

1. External CSS

External CSS is defined in a separate file with a .css extension and linked to the HTML document using the <link> element in the head section:

```
<head>
   <link rel="stylesheet" href="styles.css">
</head>
```

The external CSS file (styles.css) contains all the CSS rules:

```
body {
   font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
   line-height: 1.6;
   color: #333;
}
header {
   background-color: #4a89dc;
   color: white;
   padding: 1rem;
}
```

External CSS is the preferred method for larger projects because it:

- Separates content from presentation
- Improves caching and performance
- Enables consistent styling across multiple pages
- Simplifies maintenance

2. Internal CSS

Internal CSS is defined within the <style> element in the document's head section:

```
<head>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 0;
      padding: 0;
    }

    .container {
      width: 80%;
      margin: 0 auto;
    }
  </style>
</head>
```

Internal CSS is useful for:

- Single-page documents
- Page-specific styling that doesn't apply to other pages
- Prototyping and testing

3. Inline CSS

Inline CSS applies styles directly to individual HTML elements using the style attribute:

```
<p style="color: blue; font-size: 18px; margin-top: 10px;">
  This paragraph has inline styling.
</p>
```

Inline CSS should be used sparingly as it:

- Mixes content with presentation
- Cannot be reused across elements
- Has higher specificity that may override other styles
- Makes maintenance difficult

However, inline CSS can be useful for:

- Email templates
- One-off styling that won't be repeated
- Dynamically generated styles from JavaScript

CSS Selectors and HTML Integration

CSS selectors are patterns used to select and style HTML elements. Understanding the relationship between HTML structures and CSS selectors is crucial for effective styling:

Element Selectors

name: Select HTML elements by their tag

```
h1 {
font-size: 2.5rem;
color: #333;
}
p {
margin-bottom: 1rem;
line-height: 1.6;
}
```

Class Selectors

class attributes: Style elements with certain

```
.container {
width: 80%;
max-width: 1200px;
margin: 0 auto;
}
.btn {
display: inline-block;
padding: 0.5rem 1rem;
background-color: #4a89dc;
color: white;
border: none;
border-radius: 4px;
cursor: pointer;
}
```

ID Selectors

specific ID attributes: Page targets with

```
#header {
position: sticky;
top: 0;
z-index: 100;
background-color: white;
2px 4px rgba(0, 0, 0, 0.1);  box-shadow: 0
}
```

```
#contact-form {
padding: 2rem;
background-color: #f9f9f9;
border-radius: 8px;
}
```

Attribute Selectors

selected depending on their attributes: Syntax−Elements are

```
input[type="text"] {
padding: 0.5rem;
border: 1px solid #ddd;
border-radius: 4px;
}
a[href^="https"] {
color: green;
}
```

Combination Selectors

```
/* Descendant selector */
.article p {
font-size: 1rem;
}
/* Child selector */
.nav > li {
display: inline-block;
}
```

sibling selector = + Next

```
h2 + p {
font-weight: bold;
}
```

HTML Layout Elements

Semantic Layout Elements

HTML5 added semantic elements that give a more meaningful structure to the document:

Non-Semantic Layout Elements

Despite the use of semantic elements, these generic containers are still widely used:

CSS Layout Techniques

There are a number of layout technique that you can leverage using modern CSS that works along with your HTML structures:

Box Model

The CSS box model describes how elements are modeled as rectangular boxes:

```
.box {
width: 300px;
padding: 20px;
border: 1px solid #ddd;
margin: 20px;
margin */  margin: 0; /* Reset
}
```

Flexbox

Flexbox offers a single-dimensional layout method:

Item 1

Item 2

00C00B00000

```
.flex-container {
display: flex;
space-between;  justify-content:
align-items: center;
}
.flex-item {
flex: 1;
padding: 20px;
margin: 10px;
}
```

CSS Grid

CSS Grid offers a two-dimensional layout system:

Sidebar

Main Content

Footer

```
.grid-container {
display: grid;
1fr;  grid-template-columns: 200px
1fr auto;  grid-template-rows: auto
grid-template-areas:
"header header"
"sidebar content"
"footer footer";
```

```
min-height: 100vh;
}
. header { grid-area: header; }
{ grid-area: sidebar; }  . sidebar
content; }  . main-content { grid-area:
. footer { grid-area: footer; }
```

Responsive Layout

Web design today requires layouts that can adapt to a variety of screen sizes:

Media Queries

Media queries enable conditional application of CSS rules based on device characteristics:

```
device */  Essential styles for every
.container {
width: 90%;
margin: 0 auto;
}
/* Tablet styles */
@media (min-width: 768px) {
.container {
width: 80%;
}
}
/* Desktop styles */
@media (min-width: 1024px) {
.container {
width: 70%;
max-width: 1200px;
}
}
```

Responsive Images

Using CSS, image can be made responsive:

```
img {
max-width: 100%;
height: auto;
}
```

CSS Variables and Theming

CSS Custom Properties (variables) allow for more maintainable and themeable designs:

```
:root {
--primary-color: #4a89dc;
--secondary-color: #5d9cec;
--text-color: #333;
--background-color: #f9f9f9;
--spacing-unit: 1rem;
--border-radius: 4px;
}
.button {
core-ui-color(var(--primary-color));  background-color:
color: white;
padding: var(--spacing-unit);
var(--border-radius);  border-radius:
}
.card {
background-color: white;
color: var(--text-color);
calc(var(--spacing-unit) * 2)  padding:
border-radius );  border-radius: var( —
4px rgba(0,0,0,0.1);  box-shadow: 0px 2px
}
```

web design. When you grasp these ideas and how they are related, you will be ready to write clean, beautiful, and responsive web pages that The combination of the HTML tree structure with CSS styling via IDs, classes, and semantic elements is at the heart of modern can ensure good experiences on different devices and screen sizes.

Unit 3: Advanced HTML Features and Integration

1.7 Advanced HTML Concepts: Events, SVG, Canvas, URL Handling, and APIs in HTML5

This Module will look at the capabilities that modern HTML5 had many advanced features that enabled static HTML gives you, such as event handling, SVG, Canvas, enhanced URL handling, and native APIs for rich web applications. web pages to be dynamic, interactive applications.

Handling Events Introduction to HTML Events and

occurrences that happen in the browser which can be detected by JavaScript and that could be responded. HTML5 brought a unified Events are actions or event model that allows for interactive web experiences.

Event Attributes

HTML elements can have event attributes that run JavaScript code when events happen:

Click Me

+ this. value)" its own deadline. console.log('this changed to: ' That training data folds in>

Hover over me

Common HTML Events

Mouse Events:

- Dblclick: Fires when an element is double-clicked
- mousedown, mouseup: Fired when a mouse button is pressed and on release
- mouseover, mouseout: Raised when the mouse hovers over an element
- mousemove: Fires when mouse cursor moves on an element

Keyboard Events:

- Pressing or releasing a keykeydown,keyup
- fires when a character-producing key is pressed keypress

Form Events:

- submit: Emitted when a form is submitted
- reset: When a form is reset
- change: Fired when the value of an input element gets changed
- input: Fires when the value for an input element changes (immediate firing)

- focus, blur: Fired when an element gains focus or loses focus

Document/Window Events:

- load : The event occurs when the page is loaded.
- resize: When the browser window is resized
- scroll: Fired when the document is scrolled
- DOMContentLoaded: When the HTML document has been completely loaded and parsed

JavaScript Event Handling

While inline event attributes function, the preferred method is to attach event listeners using JavaScript:

```
// Get references to elements
const button = document. getElementById('myButton');
const form = document. getElementById('myForm');
const input = document. getElementById('myInput');
// Add event listeners
function (event) {  button. addEventListener("click",
console. log('Button clicked!' );
console. log('Event object:', event);
});
{  form. form.addEventListener('submit', function(event)
 e.preventDefault(); // Prevent form submission  event.
console. log('Form submitted!' );
});
method of responding to input events can be improved. input. However, this
console. log('Current value:', this. value);
});
longer needed  // Unload event listener when no
function handleClick() {
console. log('Temporary handler');
handleClick);  button. removeEventListener('click',
}
handleClick);  button. addEventListener("click",
```

Event Propagation

Events in HTML propagate through the DOM tree in two phases:

1. **Capturing Phase**: Events travel from the document root to the target element

2. **Bubbling Phase**: Events bubble up from the target element to the document root

```
// Third parameter determines if the listener is for capture phase (true) or bubbling phase (false, default)
parent.addEventListener('click', function() {
   console.log('Parent clicked - bubbling phase');
}, false);
parent.addEventListener('click', function() {
   console.log('Parent clicked - capturing phase');
}, true);
child.addEventListener('click', function(event) {
   console.log('Child clicked');
   event.stopPropagation(); // Prevents event from bubbling up
});
```

Custom Events

HTML5 allows creating and dispatching custom events:

```
// Create a custom event
const customEvent = new CustomEvent('userAction', {
   detail: {
      username: 'john_doe',
      timestamp: new Date()
   },
   bubbles: true,
   cancelable: true
});
// Dispatch the custom event
document.getElementById('userProfile').dispatchEvent(customEvent);
// Listen for the custom event
document.addEventListener('userAction', function(event) {
   console.log('User action detected:', event.detail);
});
```

SVG (Scalable Vector Graphics)

SVG is an XML-based markup language for creating vector graphics that scale seamlessly across different screen sizes and resolutions.

Embedding SVG in HTML

SVG can be embedded directly in HTML documents:

```
<svg width="200" height="200" viewBox="0 0 200 200">
   <circle cx="100" cy="100" r="80" fill="purple" />
```

```
  <rect x="60" y="60" width="80" height="80" fill="blue" />
  <text x="100" y="100" font-size="20" text-anchor="middle" fill="white">SVG Text</text>
</svg>
```

SVG can also be included as an external file:

```
<img src="graphic.svg" alt="Vector Graphic">
<!-- or -->
<object data="graphic.svg" type="image/svg+xml"></object>
```

Basic SVG Elements

1. **Shapes**:
    - <circle>: Creates a circle
    - <rect>: Creates a rectangle
    - <line>: Creates a line
    - <polyline>: Creates connected lines
    - <polygon>: Creates a closed shape with straight lines
    - <ellipse>: Creates an ellipse
    - <path>: Creates arbitrary paths
2. **Text**:
    - <text>: Adds text to the SVG
    - <tspan>: Adds sub-sections of text
3. **Container Elements**:
    - <g>: Groups SVG elements
    - <defs>: Defines reusable elements
    - <symbol>: Defines reusable symbols
    - <use>: Reuses elements defined elsewhere

Styling SVG

SVG elements can be styled with attributes or CSS:

```
<svg width="200" height="200">
  <style>
    .shape {
      fill: red;
      stroke: black;
      stroke-width: 2px;
    }
    circle:hover {
      fill: orange;
    }
  </style>
```

```
  <circle class="shape" cx="100" cy="100" r="80" />
</svg>
```

Interactive SVG

SVG elements can respond to events just like HTML elements:

```
<svg width="300" height="200">
  <rect x="50" y="50" width="200" height="100" fill="blue"
      onclick="this.setAttribute('fill', 'red')" />
</svg>
```

With JavaScript:

```
const svgCircle = document.querySelector('svg circle');
svgCircle.addEventListener('click', function() {
  this.style.fill = this.style.fill === 'red' ? 'blue' : 'red';
});
```

SVG Animation

SVG supports native animations with the <animate> element:

```
<svg width="200" height="200">
  <circle cx="100" cy="100" r="50" fill="blue">
    <animate attributeName="r" values="50;80;50" dur="2s"
repeatCount="indefinite" />
    <animate attributeName="fill" values="blue;purple;blue"
dur="2s" repeatCount="indefinite" />
  </circle>
</svg>
```

Canvas

The HTML5 <canvas> element provides a drawing surface for creating dynamic, scriptable 2D and 3D graphics.

Basic Canvas Setup

```
<canvas id="myCanvas" width="400" height="300"></canvas>
<script>
  const canvas = document.getElementById('myCanvas');
  const ctx = canvas.getContext('2d'); // Get 2D rendering context

  // Drawing code goes here
</script>
```

Drawing on Canvas

Canvas provides a procedural API for drawing:

```
// Set styles
```

```
ctx.fillStyle = 'blue';
ctx.strokeStyle = 'red';
ctx.lineWidth = 5;
// Draw shapes
ctx.fillRect(50, 50, 100, 75); // Filled rectangle
ctx.strokeRect(200, 50, 100, 75); // Outlined rectangle
// Draw paths
ctx.beginPath();
ctx.moveTo(50, 200);
ctx.lineTo(150, 150);
ctx.lineTo(250, 200);
ctx.closePath();
ctx.fill(); // Fill the path
ctx.stroke(); // Stroke the path
// Draw circles
ctx.beginPath();
ctx.arc(300, 200, 50, 0, Math.PI * 2); // Full circle
ctx.fill();
// Draw text
ctx.font = '20px Arial';
ctx.fillStyle = 'black';
ctx.fillText('Hello Canvas', 150, 250);
```

Canvas vs. SVG

Feature	**Canvas**	**SVG**
Rendering	Pixel-based (bitmap)	Vector-based
DOM Integration	No (single element)	Yes (each shape is an element)
Performance	Better for complex scenes	Better for fewer, larger objects
Resolution	Resolution-dependent	Resolution-independent
Accessibility	Poor (no built-in accessibility)	Good (elements can have ARIA attributes)
Animation	Manual redrawing required	CSS or SMIL animations
Event Handling	Manual hit detection required	Native event support

Canvas Animation

Canvas animations, as you may know, involve clearing and re-drawing the canvas:

50; // Starting position let x =

movement let speed=2; // Speed of

function animate() {

// Clear canvas

ctx. clearRect(0, 0, canvas. width, canvas. height);

(position object) and createa draw object which enable you to draw object at current position Filter object by position

ctx.beginPath();

ctx. arc(x, 150, 30, 0, Math. PI * 2);

ctx.fillStyle = 'green';

ctx.fill();

// Update position

x += speed;

// Bounce at edges

if (x > canvas. width - 30 || x < 30) {

speed = -speed;

}

// Continue animation

requestAnimationFrame); animate(

}

// Start animation

animate();

URL Handling in HTML5

Never ends on the way but similar event switch to address Meta Tag HTML History, history was improve with HTML5.

URL API

interface that enables you to create and manipulate URLs: The URL API is an

// Create a new URL object

= new URL(https://example.com/path/page.html?name=value const url&other=123#section);

// Access URL components

console. log(url. protocol); // "https:"

console. log(url. hostname); // "example. com"

console. log(url. pathname); // "/path/page. html"

Multiple Choice Questions (MCQs)

1. **What does HTML stand for?**
   a) Hyperlinks and Text Markup Language
   b) Hyper Text Markup Language
   c) Home Tool Markup Language
   d) Hyper Transfer Markup Language
2. **Which tag is used to create a hyperlink in HTML?**
   a) <link>
   b) <a>
   c) <href>
   d) <hyper>
3. **Which of the following is a semantic HTML tag?**
   a)
   b)
   c) <article>
   d) <b>
4. **What is the correct syntax for an HTML comment?**
   a) /* This is a comment */
   b) // This is a comment
   c) <!-- This is a comment -->
   d) ** This is a comment **
5. **Which attribute is used to open a hyperlink in a new tab?**
   a) open="new"
   b) target="_new"
   c) target="_blank"
   d) href="new_tab"
6. **What is the default alignment of table content in HTML?**
   a) Center
   b) Left
   c) Right
   d) Justify
7. **Which input type is used for selecting multiple options in a form?**
   a) <input type="text">
   b) <input type="radio">
   c) <input type="checkbox">
   d) <input type="submit">

8. **The <canvas> element in HTML is used for:**
   a) Playing audio files
   b) Rendering graphics and animations
   c) Embedding video files
   d) Structuring tabular data
9. **Which API is used in HTML5 to store data on the user's browser?**
   a) SessionStorage API
   b) WebSQL API
   c) LocalStorage API
   d) Both a and c
10. **What does the <iframe> tag in HTML do?**
    a) Embeds an external webpage within a webpage
    b) Creates a pop-up window
    c) Links to an external CSS file
    d) Defines an interactive form

Short Answer Questions

1. What is the purpose of HTML in web development?
2. Define semantic HTML and give two examples.
3. Explain the difference between <id> and <class> attributes in HTML.
4. What is the use of the <form> tag in HTML?
5. How does the <iframe> tag work, and why is it used?
6. What are the different types of input fields in an HTML form?
7. What is the difference between <table>, <thead>, <tbody>, and <tfoot>?
8. Explain the role of SVG and Canvas in HTML5.
9. What are meta tags, and why are they important?
10. How can JavaScript be integrated into an HTML file?

Long Answer Questions

1. Explain the basic structure of an HTML document with an example.
2. Discuss the significance of semantic elements in HTML and how they improve web accessibility.
3. Compare and contrast ordered lists (<ol>) and unordered lists (<ul>) with examples.

4. Describe the different types of form validation techniques in HTML5 with examples.
5. What are the differences between absolute, relative, and root-relative file paths in HTML?
6. Explain the <iframe> element in detail, with an example of how it can be used for embedding content.
7. How does the integration of CSS improve the layout and design of an HTML document? Provide examples.
8. Discuss the usage of HTML events, including onclick, onmouseover, and onchange, with examples.
9. What are the advantages of using <canvas> and <svg> for web graphics, and how do they differ?
10. Explain how HTML5 APIs such as LocalStorage and SessionStorage can be used to store data. Provide examples.

MODULE 2
CSS - Cascading Style Sheets

LEARNING OUTCOMES

By the end of this module, learners will be able to:

- Understand the purpose and types of CSS and its applications.
- Utilize different CSS selectors, including basic, advanced, and pseudo-selectors.
- Comprehend CSS specificity and inheritance rules.
- Implement background and border properties effectively.
- Manage element display and positioning using static, relative, absolute, and fixed positioning.
- Work with width, height, and overflow properties for layout control.

Unit 4: Introduction to CSS and Its Purpose

2.1 Introduction to CSS – Purpose, Types, and Application

CSS stands for Cascading Style Sheets, which was a groundbreaking technology that transformed web development forever by changing the method of designing and developing visual presentation of web content. Before CSS was introduced, HTML documents would not only define the structure of a webpage but also include presentational formatting, leading to unwieldy code that proved to be harder to maintain and less flexible when it came to design. To remedy this shortcoming, CSS was introduced, creating a greater structural separation between content (HTML) and how that content is presented (by way of CSS), creating a paradigm that lives on in the way that we develop the web today. CSS is primarily used to visually manipulate the elements of HTML globally from a single point in the code base to the whole website. There are many benefits of such separation of concerns. It firstly increases maintainability, as it enables designers to make site-wide style changes while only updating one stylesheet instead of multiple html pages. Second, CSS has a vast and significant effect on page performance. Browsers can cache style information if you use CSS, so they will be more efficient, wasting bandwidth, and enhancing the user experience. Third, it brings an unparalleled level of design flexibility and allows developers to have fine control over the layout, typography, color palette, animations, and responsive behaviors based on different screen sizes and devices. There are three types of CSS implementation, each tailored to the needs of different workflows in web development. The most recommended approach is external CSS, which means creating separate . css files that are referenced in HTML documents using tags within the document's head. This allows you to take advantage of centralized style management and browser caching. Embedded or internal CSS:Internal CSS refers to style rules placed inside tags. Inline css applies styles inline to individual HTML elements with the style attribute, giving such a high level control but losing the benefits from separation and reusability. Although inline CSS is not the best way to handle styling in large projects, it works fine for email templates or scenarios where the style needs to be scoped. CSS is being used in a multitude of real-world scenarios. One of the most

powerful and widely-used use cases is responsive web design, where we use media queries and flexible grid systems to define interfaces that will scale smoothly across a variety of screen sizes — from mobile devices to desktop displays. CSS frameworks —such as Bootstrap, Foundation, and Tailwind CSS— deliver sets of predefined components and systems for building layouts that speed up the development process while ensuring a consistent outcome. CSS preprocessors such as Sass, Less, and Stylus help enhance CSS allowing for variables, functions, and other programming constructs that improve maintainability, especially in complex projects. CSS animations and transitions allow for interactive experiences without the need for JavaScript, and CSS Grid and Flexbox offer advanced layout systems for building complex, responsive UIs with less code. As things currently stand, modern standards are being gradually standardized through the efforts of organizations such as the W3C (World Wide Web Consortium) and the CSS Working Group, detailing future developments which then make their way into implementations in browsers. Serving graceful degradation depending on the browser and its capabilities means developers can introduce modern CSS features found in CSS properties to modern browsers with fallbacks to older technology. The CSS Object Model (CSSOM) is a European Parliament (EP) API that brings dynamic style manipulation from CSS into JavaScript, the lifeblood of modern web applications. Web beyond: CSS custom properties (variables), logical properties and container queries among other, upcoming, features will take expanding the eyes of developers to the next level enabling us to implement rich, robust and performant experiences.

2.2 CSS Selectors – Basic, Advanced, and Pseudo Selectors

You rely on CSS selectors -- the basis for styling -- to apply styles to HTML elements. From general rules that apply to the entirety of a document all the way down to using hyper specific elements, understanding the full breadth of selector types gives developers ultimate control over to whom the styles will be applied. Basic selectors are a set of selectors that allow you to select an element based on intrinsic characteristics of the element's HTML. The element selector, or type selector, targets all instances of a particular HTML element type across a document. So for instance, p { color: blue; } sets blue text for all paragraph elements. This more advanced selector offers a great method for setting baseline styles for atomic HTML elements. The class selector, which is indicated by the leading period (.) is also used to select elements that share the same class attribute, allowing reusability of style under various element types. For instance,. For example, highlight { background-color: yellow; } will give a yellow background to any element that has the class "highlight." Such an approach promotes the idea of using modular, component-based styling, which is the foundation of modern CSS methodologies like BEM (Block, Element, Modifier) and SMACSS (Scalable and Modular Architecture for CSS). The ID selector begins with a hash symbol (#) and matches the only element with the corresponding value of its ID attribute. Sticks the header when scrolling past it Example: #header { position: sticky; } sticky takes effect on all element with ID of header There are many more CSS selectors available. You should not use this selector for a repeated pattern because it is associated with a unique identifier, and HTML IDs must be unique! The universal selector which is written as an asterisk (*) selects all elements in a document regardless of type. It can also be used for global resets or to set box-sizing on the entire document, like * { box-sizing: border-box; }. Although powerful, use of the universal selector should be done sparingly due to its broad scope and potential performance impact. An attribute selector allows you to apply styles to an element based on its HTML attributes, with or without specified values, allowing greater flexibility and eliminating the need for additional class or ID attributes. As an

example, input[type="text"] { border-radius: 4px; } will only give rounded corners to text input fields, allowing for the use of attribute selectors to differentiate visually or functionally similar HTML elements that use the same element type. Once we have the basic selectors down, we build on that with more advanced selectors that allow for targeting based on even more complex structures within our documents and the relationships between our HTML elements. Combinators indicate a relationship between multiple selectors, allowing for hierarchical targeting within the document. The descendant combinator (space) is used to select elements that are nested within another element, no matter how deeply nested, such as article p { line-height: 1.6; }, which selects paragraphs contained within article elements. The child combinator (>) limits the selection to direct children only, e.g. ul > li { list-style-type: square; }, affects list items that are direct children of an unordered list. The adjacent sibling combinator (+) will apply styles to an element directly following a specified element(h2 + p { font-weight: bold; })Formatting the first paragraph after any h2. All sibling combinators after some element (in the following example h3 ~ p { margin-left: 1em; }) indent all p that follow a third-level heading.

Using commas to group selectors to share the same style declaration, making your CSS more DRY and your file smaller. So, for example, h1, h2, h3 { font-family: 'Georgia', serif; } presests the same font family to all three heading levels. Compound selectors () combine multiple basic selectors without spaces between them, allowing you to target with extreme specificity. For instance, button. primary[disabled] matches disabled buttons whose class includes the "primary" keyword, so it illustrates how compound selectors can accurately select an element whose set of specific properties is a subset of an enumerable set of its attributes. There are pseudo-selectors that allow you to target specific states, positions and have something to do with dynamic or content characteristics and extend the targeting beyond the physical structure of the document. Pseudo-classes use a single colon (:) to select elements based on their state or position that exists separately from the document tree. User interaction pseudo-classes respond to user behaviors: :hover is activated when users hover the cursor over an element, :active applies while users are actively clicking or tapping, :focus highlights the

currently focused interactive element, and :visited styles links that the user has previously clicked. Structural pseudo-classes affect selection based on the document structure: :first-child and :last-child pick elements indicating that they are the first or last child of their parent, :nth-child() selects elements that fit with complex patterns or formulas (e.g. li:nth-child(odd) for alternating), :only-child matches upon determining that an element has no siblings, and finally, :empty is used to match upon elements that have no other content or children. Form-related pseudo-classes encompass the different states of form controls: :checked applies to checked checkboxes and radio buttons, :disabled and :enabled indicate the interactive state of form elements, :valid and :invalid are triggered by form validation states, and :required and :optional relate to the presence of the required attribute. Root pseudo-classes, like :root, select the highest-level element in a document body (by default) and serve as a shortcut to global CSS variables and document-wide properties. Pseudo Elements (using a double colon, ::), create and style imaginary elements that do not exist in the HTML itself. The pseudo-elements for content generation insert something before or after elements: ::before and ::after turn into virtual elements and are the first or last children of the specified element, respectively, which need content properties in order to be visible. Phrase-style pseudo elements apply their styles to small parts of the text content: ::first-letter gives style to the first letter to create that drop-cap style we all know and love, ::first-line that applies the style to the first line of text subjected to reflow upon changing viewport or resizing elements and ::selection to style the text on user selection, primarily with background and text colors. Functional pseudo-classes take arguments in parentheses, giving us parametric selection abilities: :not() removes elements that match the given selector from matching, offering negative targeting, multiple selectors can be grouped in :is() and :where() in a way that affects specificity of the group, and :has() (in newer browsers) allows selecting elements that contain certain children, providing a parent selector functionality that developers have long wished for. The correct use of selectors affect the specificity of the style and performance. Selector specificity is hierarchical, with inline styles being the most specific (1000), then IDs (100), classes and attributes (10), and elements (1). You learn about a system that decides which style will be applied

when conflicting rules apply to the same element. Although very specific selectors guarantee accurate selection, they may cause maintainability issues, introducing dependencies to concrete representations of the document. In summary, modern CSS best practices lean more toward class-based selectors, which offer a good balance between specificity and flexibility, accommodating component-based architectures that align with modern front-end development methods

.2.3 CSS Specificity and Inheritance

CSS specificity is a critical concept that defines how CSS rules will be applied when several conflicting specifications point to the same element. CSS, rather than going for the simplest possible method of the "last rule wins" variety, adopts a more nuanced calculation system that takes into account the relative importance of various selector types. Learning about specificity can go a long way towards controlling style application predictably as stylesheets start to grow more complex in large projects. In the specificity pecking order, inline styles applied to the element's style attribute are the most specific (usually as 1,0,0,0); then are ID selectors (0,1,0,0); class selectors, attribute selectors, and pseudo-classes (0,0,1,0); and finally element selectors and pseudo-elements (0,0,0,1). For compound selectors, specificity is the sum of each individual component. For instance, #sidebar p.highlight would have a specificity of 0,1,1,1 as it combines one ID, one class and one element selector. By using numbers to represent values, this provides a clear and direct system to manage style conflicts: a bigger number wins, no matter whether the CSS comes before or after it in the doc. The cascade only applies if the specificities are the same, with later rules taking precedence over earlier ones. The! This important declaration is a special case that breaks normal specificity calculations, elevating a property to the highest level. Its use is largely discouraged in professional development, both due to the maintainability that it brings as well as its use cases are generally reserved from backfilling the original classes here like if using something like utility classes or to override third-party styles. Specificity is both a blessing and a curse in CSS architecture. On the one hand, it allows accurate style application without needing to modify the document structure. On the flip-side, it can cause "specificity wars", where developers pile on selector

specificity to beat existing styles, achieving brittle, tightly coupled code which is disastrous to maintain. Modern day CSS methodologies such as BEM (Block, Element, Modifier) follow naming conventions that limit selector nesting and keeps specificity levels in check, which significantly improves CSS maintainability as well as performance. Now add css inheritance into that, inheritance works hand in hand with specificity, providing a way for style property values to flow from parent to child elements even if they are not declared. This helps to eliminate duplication of the same properties in multiple instances, as the common property can be declared once in the document tree at a higher level. Text-related properties inherit well: color, font-family, font-size, font-weight, line-height, text-align all carry to the child elements naturally from the parent. Properties such as list-style-type and list-style-position also inherit, maintaining a consistent look for your lists remains intact. Some special properties such as visibility and cursor inherit by default, and thus propagate interactive characteristics through nested elements.

However, many properties deliberately do not inherit to avoid this undesired cascading effect. Properties that affect the box model — width, height, margin, padding and border — cannot inherit, because the same dimension cannot be transferred from parent to child, otherwise, it would make any nested element unusable in a layout. Properties related to positioning such as position, top, right, bottom, and left also avoid inheritance to avoid unexpected layout shifts. Background properties (e.g. background-color and background-image) similarly do not inherit unless explicitly set to, which preserves the transparent backgrounds of nested elements, making them reveal parent backgrounds rather than duplicate them. CSS has special property values to have more explicit control over inheritance behaviors. The inherit keyword forces the property to assume the same value as its parent, which is the opposite of the default behavior of not inheriting. For example, border: inherit; makes an element inherit its parent's border style. The first keyword resets a given property value to a value for that browser (i.e., defaulted in CSS specification) prior to declaration or inheritance. Unset incorporate the behavior of both inherit and initial: it acts as inherit for inheritable properties and as initial for non-inheritable. The revert keyword (in modern browsers) is a way to back out a property to the value that is

set by the user agent stylesheet, undoing author styles but keeping what browsers would apply. This specificity versus inheritance balance is a mechanism that enables a subtle but powerful control over style application. Good CSS authors take advantage of inheritance for overall page typographic and color schemes while only applying sufficient levels of specificity to override these defaults only for particular components and states. By learning these mechanisms, developers can write predictable, maintainable stylesheets without over-duplication of rules and specificity wars. The cascade, after which CSS is named, is the third major concept in CSS, along with specificity and inheritance. It explains the order in which styles take precedence based on their origin: user agent styles (such as those set by the browser) serve as the foundation, user styles (defined at the browser settings level) take precedence over these defaults, and author styles (from website stylesheets) take precedence over both user agent and user styles. Normal declarations are processed according to specificity rules within each origin category, while! important declarations are given priority based on their origin but in reverse order:!!important user styles override important author styles and rendering accessibility accommodations not able to be overridden by website code

.**2.4 Background and Border Properties**

CSS background properties provide a powerful method for styling your element backgrounds with anything from a simple solid color to a fully layered composition with multiple images. These properties are foundational to web design: they help set visual hierarchy, provide decorative elements, and improve content readability with proper contrast. The background-color property establishes the foundational color that sits behind any other background layer, allowing you to use any valid color format including hex codes (#RRGGBB), RGB/RGBA functions, HSL/HSLA functions, and named colors. It used to be with alpha transparency a designer could apply RGBA or HSLA to create semi-transparent backgrounds, allowing content below to be partially visible creating more complex layering. The background-image property puts one or more images behind the element content and takes a URL, gradients, or multiple gradients. You can specify multiple background images in a comma-separated list, with earlier values appearing "above" later ones in the stacking order. CSS also

includes built-in gradient functions that allow the declaration of gradient images procedurally inside the css, without the need for external images, referenced with the url() function. Linear gradients built with linear-gradient() progress colors along a straight line, and use directional keywords or exact angles to determine the gradient direction. Radial gradients, created with radial-gradient(), spread outward from a center point, with parameters that dictate the shape (circle, or ellipse) and size of the gradient. Conic gradients, made with conic-gradient(), fill color around a center point and create pie-chart-like visualizations with angular color transitions. There are four background positioning properties that specify where background images are placed in their containing elements. Image positioning: the background-position property accepts keywords (top, right, bottom, left, center), percentage values, or absolute lengths, allowing for very granular definition of image position. Edge offsets (e.g. background-position: right 20px bottom 10px;), on the other hand, position images in relation to specific edges of a given container. The background-origin propertysets whether the positioning coordinates are derived from the entire element box (including borders and padding; border-box) or just the padding and content areas (padding-box, the default) or only the content area, excluding the padding (content-box). The background sizing properties determine the scaling of images with respect to their containers. The background-size property can take absolute dimensions, percentages, or special keywords: cover scales the image to fill the container completely while preserving its aspect ratio, possibly cropping pieces of the image that fall beyond the container, while contain scales the image to fit within the container completely while preserving its aspect ratio, potentially leaving empty space around the image. These keywords can be especially useful for responsive design, resizing background images automatically to meet different screen sizes and proportions. Background repetition properties control how images will fill their containers when they are smaller than the available space. The background-repeat property accepts one of the following values: repeat (tiles both left to right and up & down), repeat-x (tiles only left to right), repeat-y (tiles only up & down), no-repeat (a single instance), space (tiles additional space is added to the image without cutting off parts of tiles), or round (adjusts the tiles slightly to avoid partial tiles). The background-

attachment property controls how backgrounds are applied on scrolling: scroll (the default behavior) fixes the background relative to the element; fixed anchors the background to the viewport for pseudo parallax-like visual effects; and local attaches the background to the element's content, causing it to scroll when content overflows and scrolls within the element.

The background-clip property defines where the background is painted: border-box draws the background as far back as the borders (the default), padding-box stops the background at the inside edge of borders and content-box paints the background only into the content. The shorthand background property allows you to include all relevant background properties in a single declaration, and accepts comma-separated values for color, image position, size, repeat, attachment, origin, and clip. For multiple backgrounds, a lot of properties support comma-separated lists that correspond to each background layer. On the other hand, their playful sibling properties, border, give structures to many page layout, visually separate them, and enhance emphasis. The main border properties govern three different aspects of borders: width, style, and color. For (box) properties these absolute units (px, em, rem) and keywords (thin, medium, thick) are used to set changes between box sides: the simple border shorthand allows one to either assign equal widths for all four sides or an individual width per side if necessary. The border-style property specifies the type of border to be displayed, such as value options of solid (solid line), dashed (dashes regularly spaced), dotted (list of round dots), double (two parallel solid lines), groove (appears carved in), ridge (appears raised), inset (makes the element appear sunken) and outset (makes the element appear elevated). The border-color property defines it so it can accept any valid CSS value for colors, including transparency via RGBA or HSLA formats. Border shorthands merge multiple properties into a single declaration. The corresponding side-specific shorthands border-top, border-right, border-bottom, and border-left configure all properties for singular sides. The all-sides shorthand border sets the width, style, and color for the outside of an entire element. With shorthands, you miss giving the omitted values since they will return to the default values, making border style as much a significant property since without a declared style, borders will be invisible regardless of border width or color. Finally, border radius

properties create rounded corners, which helps soften the appearance of elements, lending itself to modern design aesthetics. The border-radius property can take absolute units or percentages, with higher values resulting in more rounding. Use corner-specific properties (border-top-left-radius, border-top-right-radius, border-bottom-right-radius, border-bottom-left-radius) to round different corners. The shorthand border-radius takes up to four values in clockwise direction, starting with the top-left, and with slash syntax (border-radius: 10px / 20px;) creates elliptical instead of circular corners. The advanced border features include images and outlines that go beyond just lines. The border-image property suite replaces standard borders with image-based borders, slicing source images into corners and sides that scale or repeat to fit the element perimeter. The below is nice for decorative aspects but it requires some extra work with the image itself as well as being aware of responsive behaviors. Many of its properties are shared with borders, although it never affects layout dimensions, adding an extra line outside of the border area around elements. This attribute is especially useful for outlines (for example, focus indicators) and transient highlights which would not interfere with adjacent content. The background and border property relationship forms the foundational scaffolding of web design elements By default, backgrounds are drawn to under borders (background-clip: border-box), so semi-transparent borders can show backgrounds underneath. The CSS box model defines how these properties work together for dimensions: the width and height properties only by default apply to the content box, with padding and borders adding to the full size of the element. The CSS declaration box-sizing: border-box changes this behavior and makes width and height include padding and borders, making it easier to calculate sizes in complex layouts. CSS layout systems (Flexbox, Grid) decide how elements that include background and borders, position themselves inside containers; transitions and animations affect these properties dynamically, to trigger interactive effects on state changes. Background and Border Properties Similar to background properties — the creative use of background and border properties helps define much of the visual character of a website. Writing layered backgrounds promote depth and texture without additional markup, helping separate content from presentation. Gradient backgrounds

increase independency on external images, improving load performance and support for dynamic color schemes by utilizing CSS variables. With more and more elements being interactive these days, border treatments are also helping to differentiate these elements, build content hierarchies, and cement brand identities through a consistent visual language. CSS has come a long way since then: from clunky solid colour backgrounds and borders, we have moved on to complex, multi-layered backgrounds and image-based borders.

2.5 Display and Positioning – Static, Relative, Absolute, Fixed

CSS display and position properties are used to control the way elements appear and how they interact on the page. So knowing these concepts very well helps the developer to build well-structured layouts with predictable results on a variety of devices and browsers. The display property defines how an element is rendered with respect to the document flow: it tells them if it is a block, inline, or one of several other display types. This property goes hand-in-hand with positioning, which determines where the elements get displayed in regard to their regular slot in the document flow—or to their parent elements.

Display Property

CSS properties since it fundamentally alters how elements behave. Some common values that The display property is quite possibly one of the most critical they include:

Block-level Block: These elements begin on a applied to all sides of, and can include block or inline elements. elements include ,, to , etc. So, it can have margin and padding, new line and extend the full width available.

Inline-block: A mixture that makes elements behave inline (next to each other) but still respects width and height as well as vertical margin and padding, gaining the advantages of both block and inline.

space and None : The element no longer takes up thus is hidden. Removes the element completely from the document flow as if it does not exist.

Flex: Makes an Element a Flex Container values, flex containers can align and distribute space among items in more sophisticated ways**.** Flex containers are the parent to flex items and provide powerful alignment controls.

Grid: Like flex, but for two-dimensional layouts instead of one-dimensional ones. So far, we had only learned about flex containers, which create either a row or a column.

Values related to tables: These are table, table-row, table-cell, etc., that make elements behave like their HTML table counterparts.

Positioning

The position property control how a node is located in the document. There are four main types:

Static Positioning

Positioned Elements

https://developer.mozilla.org/enUS/docs/Web/CSS/position#static

Positioned MakeThe top, right, bottom, left, z-index properties have no effect. This is the default for all HTML elements. Static Static:

```
.static-element {
position: static;
right, bottom, left are ignored */  /* top,
}
```

But for complex layouts or for overlapping elements it does Static positioning is simple and predictable which makes it suitable for the most at all. flowed with text. This is the positioning you have if you don't use any position property order they would do in the document. Block elements add line break before and after and in-line are The static position is when elements stack in the normal and natural not offer any flexibility.

Relative Positioning

Relative positioning positions an element relative to its normal position in the document flow. The element continues to occupy its original space, meaning other elements are not affected by its new position.

```
.relative-element {
   position: relative;
   top: 20px;
   left: 30px;
}
```

In this example, the element will be positioned 20px down and 30px to the right from where it would normally appear. Despite the visual displacement, the element still takes up space in its original position in the flow.

Relative positioning is useful for:

- Making minor adjustments to an element's position without disrupting the layout
- Serving as a positioning context for absolutely positioned child elements
- Creating a stacking context when combined with z-index

One important characteristic of relative positioning is that the element still maintains its original space in the layout. Other elements behave as if the positioned element were still in its original location, even though visually it has moved.

Absolute Positioning

Absolute positioning removes an element from the normal document flow and positions it relative to its nearest positioned ancestor (an ancestor with a position value other than static). If no positioned ancestor exists, it positions relative to the initial containing block (usually the viewport).

```
.absolute-element {
  position: absolute;
  top: 50px;
  right: 100px;
}
```

This element will be positioned 50px from the top and 100px from the right edge of its nearest positioned ancestor. The space that the element would have occupied in the normal flow is collapsed as if the element doesn't exist.

Absolute positioning has several important characteristics:

- The element is completely removed from the normal flow
- It can be positioned at specific coordinates within its positioning context
- Width defaults to fit content unless explicitly set
- It can overlap other elements without pushing them
- Other elements are arranged as if the absolute element doesn't exist

Absolute positioning is particularly useful for:

- UI elements that need to be precisely placed, like modals or tooltips
- Elements that need to appear on top of other content
- Creating complex layouts with overlapping elements

- Creating dropdown menus
- Custom styled form elements

When using absolute positioning, developers must be careful to ensure the element remains visible within its container and doesn't overlap important content unintentionally.

Fixed Positioning

Fixed positioning is similar to absolute positioning, but the element is positioned relative to the viewport (browser window) rather than any ancestor element. This means the element stays in the same place even when the page is scrolled.

```
.fixed-element {
  position: fixed;
  bottom: 20px;
  right: 20px;
}
```

This example shows an element that will appear 20px from the bottom and 20px from the right of the viewport, remaining in that exact position regardless of scrolling.

Fixed positioning is commonly used for:

- Navigation bars that stay at the top or side while scrolling
- Back-to-top buttons
- Cookie consent banners
- Chat widgets or helpdesk buttons
- Floating action buttons on mobile interfaces

Like absolutely positioned elements, fixed elements are removed from the document flow and can overlap other content. This can sometimes cause usability issues on smaller screens if not implemented carefully.

Sticky Positioning

Although not explicitly mentioned in the section title, sticky positioning is worth mentioning as it combines aspects of both relative and fixed positioning. An element with position: sticky behaves like a relatively positioned element until it crosses a specified threshold, at which point it behaves like a fixed element.

```
.sticky-element {
  position: sticky;
  top: 0;
}
```

This creates an element that scrolls normally with the page until it reaches the top of the viewport, at which point it "sticks" and remains visible at the top as the rest of the content continues to scroll.

Sticky positioning is useful for:

- Section headers in long lists
- Navigation elements that should remain visible during scrolling
- Table headers that should remain visible when scrolling through table data
- Sidebar elements that should remain in view

Browser support for sticky positioning has improved significantly, making it a reliable option for modern websites.

Z-Index and Stacking Context

When elements overlap due to positioning, the z-index property determines which elements appear on top. Elements with higher z-index values appear in front of elements with lower values.

```
.back-element {
   position: absolute;
   z-index: 1;
}
.front-element {
   position: absolute;
   z-index: 2; /* This will appear on top */
}
```

It's important to note that z-index only works on positioned elements (elements with position set to something other than static). Additionally, z-index creates stacking contexts, which can limit the scope of z-index comparisons to specific parent-child relationships.

Common Positioning Patterns and Use Cases

Centering Elements

Absolute positioning can be used to center elements both horizontally and vertically:

```
. centered {
position: absolute;
top: 50%;
left: 50%;
-50%);  transforms: translate(-50%,
}
```

This technique places the element at 50% the top and left of its containing element and then uses the transform property to shift it back half its own width and height, centering it perfectly.

Overlays and Modals

To create overlay elements like modals, fixed or absolute positioning is usually used:

```
.overlay {
position: fixed;
top: 0;
left: 0;
width: 100%;
height: 100%;
0, 0, 0.5);  style background-color: rgba(0,
z-index: 10;
}
.modal {
position: fixed;
top: 50%;
left: 50%;
translate(-50%,-50%);  transform:
z-index: 11;
}
```

Sticky Headers and Footers

Fixed positioning creates headers and footers that stay on screen while the user scrolls:

```
.header {
position: fixed;
top: 0;
width: 100%;
z-index: 100;
}
.footer {
position: fixed;
bottom: 0;
width: 100%;
}
```

Floating Action Buttons

Fixed positioning is also perfect for floating action buttons that are usually found in mobile interfaces:

Considerations Best Practices

```
action-button {
position: fixed;
bottom: 20px;
right: 20px;
width: 60px;
height: 60px;
border-radius: 50%;
background-color: #2196F3;
}
```

Responsive Design Challenges

In responsive design the fixed and absolute positioning can create some issues, as the elements with such positioning is removed from normal for positioning Instead of pixel values, use percentage values to reposition elements for various viewports flow and are not responsive means, it does not adjust itself as you change viewport sizes. Solutions include:

- Applying media queries
- Relative positioning
- Applying different positioning methods depending on the viewport size
- (vw, vh) based on viewport units

Performance Considerations

So for a limited number of elements, Positioned elements (specifically those with fixed positioning) can degrade performance because they cause certain parts of a page to be rendered as different layers in the that's not a worry, but it can degrade performance for complex layouts with lots of positioning elements. browser's rendering engine.

Accessibility Implications

Developers need to make sure their content stays accessible when using positioning to create unique layouts. This includes:

keyboard navigation

- Logical tab order for

important content

- Making sure fixed elements don’t cover

work around fixed elements

- Offering mechanisms to dismiss or

to make sure the content stays legible

- Using screen readers

2.6 Width, Height, and Overflow Properties

Width and height determine the physical size of elements (and overflow properties help us deal with the text that overflows these Managing the size of elements is a basic principle of CSS that affects layout, user interface, dimensions).

Width and Height Properties

CSS offers various approaches to setting elements' width and height, each applicable to particular situations and functioning in its own manner.

Basic Width and Height

With specific values, the most basic way is to use the width and height properties:

```
.box {
width: 300px;
height: 200px;
}
```

These properties may take different units:

Pixels (px): Static-size units that render the same across devices, making them ideal for elements that need to retain a precise size, regardless of the context.

units (vw, vh): Relative to the viewport size, where 1vw = 1% of viewport width, and 1vh = 1% of viewport height. This is especially handy for responsive designs that Viewport adapt to varying screen sizes.

These are often used em and rem These units should be understood relative to font size and are the basis of responsive typography, with the difference that em is relative to the font size of the element and rem is used relative for typography, but can come in handy for making a scalable layout. to the font size of the root element.

Min and Max Dimensions

CSS has min-width, max-width, min-height, and max-height properties that set limits instead of specific sizes:

```
.responsive-element {
width: 100%;
max-width: 800px;
min-height: 200px;
}
```

min-width/min-height: Which prevents things from getting too small to for responsive design This property is crucial be usable or look good on small screens.

max-width/max-height: Limit how much space your element can occupy on a particularly wide screen which would increase readability or maintain the intended design.

width: 100%; max-width: X, which lets an element fill its container up to a certain size. When it comes to responsive design this is a very common pattern:

Box-Sizing Property

The box-sizing property has a big impact on how width and height are calculated:

```
.content-box {
*/  box-sizing: content-box; /* Default
width: 300px;
padding: 20px;
border: 1px solid black;
342px */ / Total fucking width: 300px + 40px (padding) + 2px (border) =
}
.border-box {
box-sizing: border-box;
width: 300px;
padding: 20px;
border: 1px solid black;
border.  Total width: 300px (includes padding and
}
```

The content-box: The default, where width and height only includes the content padding and border are added to the defined heights.

Making it more intuitive and easier to predict border-box: how they will size, since what you can see, will align with the size you would specify. your content, padding, and borders.

Width and height include

Box model Many developers use the universal selector to set box-sizing: border-box on all elements for more consistent layouts:

```
*, *::before, *::after {
box-sizing: border-box;
}
```

Auto Width and Height

When width or height is auto inline-block elements,collapses to fit the content. of the container. For inline and width; auto; If the element is a block element, it will take the full width to the height of the content, showing all the content. height:auto: Resizes according (the default for many elements), the browser determines the appropriate dimension:

Aspect Ratio

Modern CSS includes support for the aspect-ratio property, which preserves a certain ratio of width to height:

```
.video-container {
width: 100%;
aspect-ratio: 16 / 9;
}
```

It is especially useful in the case of responsive media elements and helps to preserve the ratio across screens.

Overflow Properties

This is useful when a width and height are set, leading to text larger than or extending out The overflow property specifies how to handle content that is too big to fit of these limitations. into an area.

Basic Overflow

values: The overflow property can have multiple

visible: Content will not be clipped and may be rendered outside the box of the element.

hidden: The content that might extend beyond the element is hidden.

scroll: Scrollbars show up regardless of whether the content A scrollbar will be overflows. added to the element so the user can scroll to see the overflowing content.

auto: Scrollbars are only displayed when the content is overflowing the boundaries of the element

```
.text-container {
width: 300px;
height: 200px;
```

```
overflow: auto;
}
```

Directional Overflow

:

```
.table-container {
width: 100%;
max-height: 400px;
necessary */  overflow-x: auto; /* Allows horizontal scrolling if
/* Disable vertical scrolling */  overflow-y: hidden;
}
```

This can come in handy especially for tables, codeblocks, or other content that you might want scrolling on the horizontal axis (while still keeping constraints on the vertical).

Text Overflow

For text only overflow control, the text-overflow property adds more value:

```
.truncated-text {
width: 200px;
white-space: nowrap;
overflow: hidden;
text-overflow: ellipsis;
}
```

To truncate multi-line text, we need a set needed with UI elements where space is at a premium. This produces text that truncates with a series of ellipsis (…) if the text longer than the width of its container, as is often of properties:

```
.multiline-truncated {
width: 300px;
/* ~ 3 lines of text */  max-height: 4.5em;
overflow: hidden;
display: -webkit-box;
-webkit-line-clamp: 3;
-webkit-box-orient: vertical;
}
```

Scroll Behavior

The scroll-behavior property determines how programmatic scrolling animations work:

```
.smooth-scroll {
```

```
height: 300px;
overflow: auto;
scroll-behavior: smooth;
}
```

This allows for smooth scroll animations when users navigate within the element using JavaScript or anchors.

Practical Applications

Responsive Images

Responsive control of image dimensions while maintaining aspect ratios:

```
.responsive-image {
max-width: 100%;
height: auto;
}
```

This simple pattern prevents images from exceeding the width of their container while keeping their original aspect ratio.

Card Components

Fixed-height cards and controlled overflow for same UI:

```
.card {
width: 300px;
height: 400px;
}
.card-header {
height: 60px;
}
.card-image {
height: 200px;
overflow: hidden;
}
.card-content {
height: 140px;
overflow: auto;
}
```

Modal Windows

content for modals with large amount of content Scrollable

```
.modal {
position: fixed;
top: 50%;
```

```
left: 50%;
absolute;  position:
width: 80%;
max-width: 600px;
max-height: 80vh;
overflow: auto;
}
```

Horizontal Scrolling Sections

Making horizontal scrolling sections in the likes of a gallery or feature:

```
.horizontal-scroll {
width: 100%;
overflow-x: auto;
white-space: nowrap;
}
.horizontal-scroll.item {
display: inline-block;
width: 250px;
height: 300px;
margin-right: 20px;
}
```

Best Practices and Considerations

Responsive Design

In responsive designing fixed dimensions can give you a headache across multiple screen sizes. Instead:

- Use relative units: percentages and viewport units
- Use max-width and min-width constraints instead of fixed widths
- Use content adaptive heights (height: auto)
- Adjust dimensions according to viewport size with media queries

Performance Considerations

Overflow properties can affect performance, specifically with the use of when applied in large or many éléments:

- Scrolling Containers for example, can by their nature be performance taxing on mobile
- Avoid very long scrolling containers use pagination instead
- use overflow: auto instead of overflow scroll to avoid unwanted scrollbars

- Use nested scrolling containers with care, as they can result in challenging user experiences

Accessibility Implications

Any overthrow controls also need to be added with accessibility in mind:

- Make scrollable areas obvious to every user
- Get keyboard access to scrollable content
- Don't hide content users might need to scroll to access without indicators
- Run screen readers to make certain that relationships between content are still clear

2.7 List Styles and calc() Function

List Styles

HTML lists are fundamental elements for presenting related items in a structured format. CSS provides extensive styling capabilities for lists through various properties and techniques.

Basic List Properties

CSS offers several properties specifically for styling lists:

- **list-style-type**: Defines the marker (bullet or numbering) style for list items.
- **list-style-position**: Determines whether the marker appears inside or outside the content flow.
- **list-style-image**: Allows using a custom image as the list marker.
- **list-style**: A shorthand property combining the above three properties.

List Style Types

The list-style-type property offers numerous predefined marker styles:

For unordered lists (<ul>):

- **disc**: The default filled circle
- **circle:** An empty circle
- **square:** A filled square
- **none:** No marker

For ordered lists (<ol>):

- **decimal:** Standard numbers (1, 2, 3)
- **decimal-leading-zero:** Numbers with leading zeros (01, 02, 03)

- **lower-roman:** Lowercase Roman numerals (i, ii, iii)
- **upper-roman**: Uppercase Roman numerals (I, II, III)
- **lower-alpha or lower-latin:** Lowercase letters (a, b, c)
- **upper-alpha or upper-latin:** Uppercase letters (A, B, C)
- **lower-greek:** Lowercase Greek letters (α, β, γ)

```
ul {
   list-style-type: square;
}
ol {
   list-style-type: upper-roman;
}
```

In addition to these common values, CSS also supports more specialized numbering systems like armenian, georgian, and various language-specific numbering via cjk-ideographic (Chinese-Japanese-Korean), hiragana, katakana, and others.

List Style Position

The list-style-position property controls whether the marker appears inside or outside the content block:

```
ul {
   list-style-position: outside; /* Default */
}
ol {
   list-style-position: inside;
}
```

outside: Places markers in the margin area, creating a cleaner alignment of the actual content.

inside: Places markers within the content area, which can be useful when you need the entire list, including markers, to fit within a specific width.

Custom List Markers with Images

For more distinctive list styling, custom images can replace standard markers:

```
ul {
   list-style-image: url('bullet.png');
}
```

However, this basic approach offers limited control over marker size and positioning. For more precise control, many developers prefer using background images or pseudo-elements instead.

List Style Shorthand

The list-style shorthand property combines type, position, and image in a single declaration:

```
ul {
   list-style: square inside url('bullet.png');
}
```

The order doesn't matter, and any values can be omitted, with defaults applied for missing values.

Advanced List Styling Techniques

While basic list properties provide a foundation, more sophisticated styling often requires additional techniques.

Styling with Pseudo-elements

Using : before pseudo-elements offers precise control over marker appearance:

```
ul {
   list-style-type: none; /* Remove default markers */
   padding-left: 1.5em; /* Space for custom markers */
}
ul li::before {
   content: "→ "; /* Custom marker */
   display: inline-block;
   width: 1em;
   margin-left: -1em;
   color: #0066cc;
}
```

This approach allows complete customization of marker color, size, spacing, and even animations or transitions.

Multi-level Lists

For nested lists, different marker styles can visually distinguish hierarchy levels:

```
ul {
   list-style-type: disc;
}
ul ul {
   list-style-type: circle;
}
ul ul ul {
   list-style-type: square;
```

```
}
```

Horizontal Lists

Lists can be transformed into horizontal navigations or button groups:

```
ul.horizontal {
   list-style-type: none;
   padding: 0;
   margin: 0;
}
ul.horizontal li {
   display: inline-block;
   margin-right: 1em;
}
```

For more sophisticated horizontal lists, flexbox provides additional control:

```
ul.flex-horizontal {
   list-style-type: none;
   padding: 0;
   margin: 0;
   display: flex;
   gap: 1em;
}
```

Custom Counters

For complex numbering requirements, CSS counters offer programmatic control over numbering:

```
ol {
   list-style-type: none;
   counter-reset: item;
}
ol li {
   counter-increment: item;
}
ol li::before {
   content: "Section " counter(item) ": ";
   font-weight: bold;
}
```

Counters can be nested, formatted, and combined with text or other content, making them powerful for document structuring.

Practical List Styling Examples

Modern Navigation Menu

```
.nav-menu {
   list-style-type: none;
   padding: 0;
   margin: 0;
   display: flex;
   gap: 20px;
}
.nav-menu li a {
   text-decoration: none;
   color: #333;
   font-weight: 500;
   transition: color 0.3s;
}
.nav-menu li a:hover {
   color: #0066cc;
}
```

Feature List with Custom Icons

```
.feature-list {
   list-style-type: none;
   padding: 0;
}
.feature-list li {
   padding-left: 2em;
   margin-bottom: 1em;
   position: relative;
}
.feature-list li::before {
   content: "✓";
   position: absolute;
   left: 0;
   color: #4CAF50;
   font-weight: bold;
}
```

Timeline or Process Steps

```
.timeline {
   list-style-type: none;
```

```
    padding: 0;
    position: relative;
}
.timeline::before {
    content: "";
    position: absolute;
    top: 0;
    bottom: 0;
    left: 15px;
    width: 2px;
    background: #ddd;
}
.timeline li {
    padding-left: 40px;
    position: relative;
    margin-bottom: 30px;
}
.timeline li::before {
    content: "";
    position: absolute;
    left: 10px;
    top: 5px;
    width: 12px;
    height: 12px;
    border-radius: 50%;
    background: #0066cc;
    border: 2px solid white;
}
```

The calc() Function

The calc() function is a powerful CSS feature that allows mathematical calculations for property values, enabling dynamic and responsive layouts with precision.

Basic Syntax and Operations

The calc() function performs calculations using standard mathematical operators:

- Addition (+)
- Subtraction (-)
- Multiplication (*)

- Division (/)

```
.element {
  width: calc(100% - 40px);
}
```

This example sets the width to 100% of the parent container minus 40 pixels, creating responsive padding without additional elements.

Important syntax rules include:

- Spaces must surround + and - operators
- Spaces are optional for * and / operators
- Calculations can be nested within parentheses
- Values can mix different units

Mixing Units

One of calc()'s most powerful features is combining different unit types in a single calculation:

```
.column {
  width: calc(50% - 20px);
  margin-left: calc(1rem + 2vw);
  height: calc(100vh - 5em);
}
```

This allows for truly responsive designs that combine the benefits of relative (%, em, rem, vw) and absolute (px, pt) units.

Common Use Cases

Fluid Typography

```
body {
  font-size: calc(16px + 0.5vw);
}
```

This creates text that scales smoothly with viewport width without requiring media queries.

Flexible Layouts with Fixed Elements

```
.sidebar {
  width: 250px;
}
.main-content {
  width: calc(100% - 250px);
}
```

This maintains a fixed-width sidebar while allowing the main content to use the remaining space.

Centering with Fixed Margins

```
.container {
   width: calc(100% - 40px);
   max-width: 1200px;
   margin: 0 auto;
}
```

This centers a container with consistent 20px margins on each side while respecting a maximum width.

Grid-like Layouts

```
.grid {
   display: flex;
   flex-wrap: wrap;
}
.grid-item {
   width: calc((100% - 40px) / 3);
   margin-right: 20px;
   margin-bottom: 20px;
}
.grid-item:nth-child(3n) {
   margin-right: 0;
}
```

This creates a three-column grid with 20px gutters without requiring CSS Grid or complex calculations.

Variable-height Elements

```
.section {
   min-height: calc(100vh - 80px); /* Subtracting header height */
}
```

This ensures sections fill the viewport minus the header height.

Nested Calculations

Calc() functions can be nested for more complex scenarios:

```
.element {
   width: calc(50% - calc(20px + 2%));
}
```

This can be simplified to:

```
.element {
   width: calc(50% - 20px - 2%);
}
```

Variables and calc()

Calc() works seamlessly with CSS variables for powerful, maintainable layouts:

```
:root {
   --gutter: 20px;
   --columns: 3;
}
.grid-item {
   width: calc((100% - (var(--gutter) * (var(--columns) - 1))) / var(--columns));
   margin-right: var(--gutter);
}
.grid-item:nth-child(var(--columns)n) {
   margin-right: 0;
}
```

This allows changing the grid structure by updating variables rather than rewriting calculations.

Browser Support and Limitations

Modern browsers have excellent support for calc(), but some considerations include:

- Avoid division by zero or potentially zero values
- Complex calculations can impact rendering performance
- Very old browsers (IE8 and earlier) don't support calc()
- Some properties may have unexpected behavior with calc() in certain browsers

When using calc() in production, always test across target browsers and provide fallbacks where necessary.

2.8 Visibility and Print-Specific CSS

Visibility Properties

CSS provides several methods to control element visibility, each with distinct behaviors and use cases. Understanding these differences is crucial for creating interactive interfaces and responsive designs.

display: none vs. visibility: hidden

The two primary methods for hiding elements have important differences:

display: none

```
.hidden-display {
   display: none;
```

```
}
```

When an element has display: none:

- It's completely removed from the document flow
- It takes up no space in the layout
- It's not visible to screen readers or assistive technology
- Child elements cannot override this and become visible
- It's not part of the tab order
- It will not receive mouse events or focus

This is appropriate for:

- Content that should be completely removed from the page until needed
- Implementing "toggle" features where space allocation changes when elements appear/disappear
- Initial states for elements that will be revealed by JavaScript

visibility: hidden

```
.hidden-visibility {
   visibility: hidden;
}
```

When an element has visibility: hidden:

- It remains in the document flow
- It takes up space in the layout as if it were visible
- It's not visible to users but may still be read by some screen readers
- Child elements can override this with visibility: visible
- It's not part of the tab order
- It will not receive mouse events

This is appropriate for:

- Elements that should maintain their space in the layout when hidden
- Situations where you need to measure hidden elements
- When you need to selectively show child elements of a hidden parent

Opacity and Alpha Transparency

Another approach to controlling visibility is adjusting transparency:

```
.transparent {
   opacity: 0.5;  /* 50% transparency */
}
.invisible {
```

```
    opacity: 0;    /* Completely transparent but still present */
}
.transparent-background {
    background-color: rgba(255, 0, 0, 0.5);  /* Red with 50% alpha */
}
```

When an element has opacity: 0:

- It remains in the document flow
- It takes up space in the layout
- It can receive mouse events and focus
- It's part of the tab order by default
- All child elements inherit the transparency

This approach is useful for:

- Fade in/out animations and transitions
- Hover effects
- Interactive elements that should respond even when not visible
- Creating overlay effects

Clip and Clip-path

For more complex hiding requirements, clipping can be used:

```
.clipped {
    position: absolute;
    clip: rect(0, 0, 0, 0);
}
.clip-path {
    clip-path: circle(0);
}
```

These techniques:

- Remove the element visually while keeping it in the DOM
- Can be useful for accessibility patterns where elements should be available to screen readers but not visible
- Allow for creative reveal animations when combined with transitions

The modern approach uses clip-path which offers more flexibility and animation capabilities than the older clip property.

Hidden Attribute

HTML5 introduced the hidden attribute as a semantic way to hide elements:

```
<div hidden>This content is hidden</div>
```

This has similar effects to display: none but carries semantic meaning. It can be overridden with CSS:

```
[hidden] {
   display: block !important;
}
```

Combining Methods for Custom Visibility Patterns

Different techniques can be combined for specific requirements:

Accessible Hidden Content (Visually hidden but screen reader accessible)

```
.visually-hidden {
   position: absolute;
   width: 1px;
   height: 1px;
   padding: 0;
   margin: -1px;
   overflow: hidden;
   clip: rect(0, 0, 0, 0);
   white-space: nowrap;
   border: 0;
}
```

This pattern hides content visually while keeping it accessible to screen readers essential for implementing proper accessibility.

2.9 Cursor and Button Styling

The visual feedback provided by cursors and buttons plays a crucial role in creating intuitive and responsive user interfaces. These elements serve as the primary interaction points between users and websites, making their styling essential for effective user experience design.

Cursor Styling

CSS provides the cursor property to modify how the mouse pointer appears when hovering over different elements. This subtle yet powerful feature helps communicate to users what actions are possible.

```
.clickable {
  cursor: pointer; /* Changes to a hand icon indicating clickability */
}
.text-selection {
  cursor: text; /* Changes to text selection I-beam */
```

```
}
.draggable {
  cursor: move; /* Changes to a move icon */
}
.loading {
  cursor: wait; /* Changes to loading indicator */
}
.disabled {
  cursor: not-allowed; /* Indicates an element cannot be interacted with */
}
```

Custom cursors can also be implemented for more unique interfaces:

```
.custom-cursor {
  cursor: url('path/to/custom-cursor.png'), auto;
}
```

Button Styling

Buttons are among the most common interactive elements on websites, and their styling can significantly impact user engagement. Effective button styling involves several key aspects:

Basic Button Styling

```
.button {
  display: inline-block;
  padding: 10px 20px;
  background-color: #3498db;
  color: white;
  border: none;
  border-radius: 4px;
  font-family: 'Arial', sans-serif;
  font-size: 16px;
  text-align: center;
  text-decoration: none;
  cursor: pointer;
}
```

Interactive States

Well-designed buttons provide visual feedback for different interaction states:

```
.button:hover {
  background-color: #2980b9; /* Darker shade when hovered */
```

```
}
.button:active {
  background-color: #1f6aa5; /* Even darker when clicked */
  transform: translateY(1px); /* Slight movement to simulate pressing */
}
.button:focus {
  outline: 2px solid #74b9ff; /* Accessibility feature for keyboard navigation */
  outline-offset: 2px;
}
.button:disabled {
  background-color: #cccccc;
  color: #999999;
  cursor: not-allowed;
  opacity: 0.7;
}
```

Button Variations

Different types of buttons can be styled distinctly to communicate their purpose:

```
/* Primary action button */
.button-primary {
  background-color: #3498db;
}
/* Secondary action button */
.button-secondary {
  background-color: transparent;
  border: 2px solid #3498db;
  color: #3498db;
}
/* Danger action button */
.button-danger {
  background-color: #e74c3c;
}
/* Success action button */
.button-success {
  background-color: #2ecc71;
}
```

Button Sizes

different contexts. Providing multiple button sizes gives you flexibility in

```
.button-small {
padding: 5px 10px;
font-size: 12px;
}
.button-large {
padding: 15px 30px;
font-size: 18px;
}
```

Modern Button Effects

add visual interest: Modern button styles use subtle effects to

```
.button-modern {
transition: all 0.3s ease;
0, 0, 0.10); box-shadow: 0 2px 4px rgba(0,
}
.button-modern:hover {
transform: translateY(-2px);
```

0 6px 8px rgba(0, 0, 0, 0.15); Hold you also as opaque and without respect to their end up appearing as a result when your cursor under the following defining a special effect for box-shadow:

```
}
.button-modern:active {
transform: translateY(0);
0, 0, 0.1); box-shadow: 0 2px 4px rgba(0,
}
```

Unit 6: Advanced Styling Techniques and Responsive Design

2.10 Advanced CSS Topics: Images, Colors, Gradients, Shadows, Fonts, Transformations, Animations, and Z-Index

can add up to a unified and logical UI. with the elements(interaction states). These small details When you spend time on cursor/button styling, it adds some beauty to your website and makes it more user-friendly by giving clear hints on where the user is interacting Fonts, Transformations, Animation and Z-Index Other Elements in CSS: Images, Colors, Gradients, Shadows,

Image Handling

CSS provides several properties to adjust how images are displayed and integrated inside a layout.

Basic Image Properties

```
.responsive-image {
*/  max-width: 100%; /* Allows the image to be responsive
actually true.  None of the statements above are
}
.background-image {
url(relative/path/to/image. jpg');  background-image:
the entire container */  background-size: cover; /* covers
image */  background-position: center; /* center
no-repeat; /* Do not tile */  background-repeat:
}
```

Image Filters

CSS filters can perform advanced image manipulations:

```
.filtered-image {
grayscale(100%); /* Brews it in black and white (maybe darker) */
filter:
}
.filtered-image:hover {
*/ filter: grayscale(100%); /* Grayscale/black and white, color on
hover
ease 0.5s; /* Smooth transition */  transition: filter
}
.multiple-filters {
sepia(40);  filter: contrast(100) brightness(90)
```

```
}
```

Object-Fit Property

The property object-fit defines how an image or video should be resized to fit its container:

```
.contain-image {
width: 300px;
height: 200px;
*/  object-fit: contain; /* Maintain aspect ratio within mega
}
.cover-image {
width: 300px;
height: 200px;
entire box, may get cropped */  object-fit: cover; /* Covers
}
```

Advanced Color Techniques

options and opacity control is available. With modern CSS, color format

Color Formats

```
.color-examples {
#3498db; /* Hexadecimal */  color-1:
HEX */  color-2: #3498db; /*
0.7); /* RGBA */  color-3: rgba(52, 152, 219,
189); /* RGB */  color-4: rgb(49, 130,
HSLA with alpha */  color-5: hsla(204, 70%, 53%, 0.7); /*
}
```

Color Variables

CSS custom properties (variables) let us centralize the management of colors:

```
:root {
--primary-color: #3498db;
--secondary-color: #2ecc71;
--text-color: #333333;
--accent-color: #e74c3c;
}
.element {
color: var(--text-color);
background-color:var(--primary-color);}
  border: 2px solid var(--accent-color);
```

```
}
```

Gradients

Gradients create smooth transitions between colors, adding depth and visual interest.

Linear Gradients

```
.linear-gradient {
  background: linear-gradient(to right, #3498db, #2ecc71);
}
.multi-stop-gradient {
  background: linear-gradient(45deg, #3498db, #2ecc71, #e74c3c);
}
.transparent-gradient {
  background: linear-gradient(to bottom, rgba(52, 152, 219, 1), rgba(52, 152, 219, 0));
}
```

Radial Gradients

```
.radial-gradient {
  background: radial-gradient(circle, #3498db, #2ecc71);
}
.positioned-radial {
  background: radial-gradient(circle at top right, #3498db, #2ecc71);
}
```

Conic Gradients

```
. conic-gradient {
#ff5161, #7f1b43, #ff6988);  background: conic-gradient(from 45deg, #ff6988,
wheel */  border-radius: 50%; /* Makes it appears as a color
}
```

Shadows

Shadows help create depth and dimension to make interfaces more visually interesting.

Box Shadows

```
.basic-shadow {
rgba(0,0,0,0.1);  box-shadow: 0 4px 8px
}
.multiple-shadows {
box-shadow:
0 2px 4px rgba(0, 0, 0, 0.1),
```

```
0, 0.1);  0px 8px 16px rgba(0, 0,
}
.inset-shadow {
0.1);  box-shadow: inset 0 2px 4px rgba(0, 0, 0,
}
.colored-shadow {
62, 80, 0.5);  box-shadow: 0 4px 8px rgba(44,
}
```

Text Shadows

```
.basic-text-shadow {
1px 1px 2px rgba(0, 0, 0, 0.3);  text-shadow:
}
.glow-effect {
.8);  text-shadow: 0 0 10px rgba(52, 152, 219,
}
.multiple-text-shadows {
text-shadow:
1px 1px 2px #3498db,
-1px -1px 2px #e74c3c;
}
```

Font Styling

Typography is a central aspect of design, and CSS provides deep control over text formatting.

Web Fonts

```
@font-face {
font-family: 'CustomFont';
src: url('fonts/custom-font. woff2') format('woff2'),
url('fonts/custom-font. woff') format('woff');
font-weight: normal;
font-style: normal;
loaded */  font-display: swap; /* How font is
}
.custom-font-text {
CSS: font-family: 'CustomFont', sans-serif;  Or, as really scripted in
}
```

Variable Fonts

```
@font-face {
font-family: 'VariableFont';
```

```
format('woff2-variations');  url('fonts/variable-font.eot'); woff2')  src:
100 900; /* Weight range */  font-weight:
}
.variable-font-example {
sans-serif;  font-family: 'VariableFont',
/* Any value between 100-900 */  Font-weight: 275;
*/  font-variation-settings: 'wght' 275, 'wdth' 80; /* More axes
}
```

Advanced Text Styling

```
.stylized-text {
font-variant: small-caps;
letter-spacing: 1.5px;
line-height: 1.6;
text-transform: uppercase;
underline wavy #3498db;  `text-decoration:
text-underline-offset: 5px;
font-kerning: normal;
"liga" on, "kern" on;  font-feature-settings:
}
```

Transformations

CSS transforms change how elements look on the page, but do not change where they live (flow) in the document.

2D Transforms

```
.translate-example {
element right 20px  transform: translate(20px, 30px); /Move & down
30px/
}
.rotate-example {
*/  transform: rotate(360deg); /* complete rotation
}
.scale-example {
original size! */  transform: scale(1.5); /* 150% the
}
.skew-example {
both X and Y */  transform: skew(10deg, 5deg); /* skewing on
}
.combined-transform {
scale(1.2);  transform: translate(20px, 0) rotate(45deg)
```

```
}
```

3D Transforms

```
.perspective-container {
Set the space into 3D */  perspective:1000px; /*
}
.rotate3d-example {
have transform: rotateX(45deg) rotateY(30deg) rotateZ(15deg);  you
preserve-3d; /* 3D positioning of children is preserved */  transform-
style:
}
.flip-card {
transform: rotateY(180deg);
rotate */ backface-visibility: hidden; /* makes the back face hidden
when
}
```

Animations

CSS animations enable smooth transitions between states to create dynamic, interactive user interfaces.

Keyframe Animations

```
@keyframes slideIn {
0% {
transform: translateX(-100%);
opacity: 0;
}
100% {
transform: translateX(0);
opacity: 1;
}
}
.animated-element {
forwards;  animation: inFromRight 1s ease-out
}
```

Animation Properties

```
.customized-animation {
animation-name: slideIn;
animation-duration: 1.5s;
ease-in-out;  transition-timing-function:
animation-delay: 0.2s;
```

```
animation-iteration-count: 2;
backwards;  animation-direction:
backards;  animation-fill-mode:
running;  animation-play-state:
}
```

Multiple Animations

```
.multi-animated {
animation:
slideIn 1s ease-out,
fadeColor 3s linear infinite;
}
```

Transitions

elements. Triggers are helpful for state changes on interactive

```
.transition-example {
background-color: #3498db;
ease, transform 500ms ease-out;  transition: background-color 300ms
}
.transition-example:hover {
background-color: #2ecc71;
transform: scale(1.1);
}
. staggered-transition. item {
}
*.05s));  transition-delay: calc((var(--item-index)
}
```

Z-Index and Stacking Context

The z-index is a CSS property that decides which element stays over another.

```
.stacking-example {
to make z-index work */  position: relative; /* Needed
on top */  z-index: 10; /* Higher value comes
}
.stacking-context {
context  Create a new stacking
position: relative;
z-index: 0;
stacking context as well */  opacity: 0.99; /* Creates a
}
```

is understanding stacking contexts: A vital part of controlling layered layouts

```
.parent {
position: relative;
z-index: 1;
}
.child {
position: absolute;
Only matters compared to this parent’s stacking context */ z-index: 100; /*
}
.sibling {
position: relative;
and all its children */ be on top of. parent z-index: 2; /* Will
}
```

Once you get the hang of them, you can create quite sophisticated designs that actually Some of these advanced CSS topics can be powerful improve user experience without sacrificing performance or accessibility.tools for visually appealing and interactive websites.

2.11 Responsive Web Design – CSS Media Queries

At Responsive web design: Responsive web design is one of the important techniques the core of this approach lies the CSS media queries which allow developers to specify styles based on device attributes. used for web development and aims at making the website readable and functional across multiple devices and screen sizes.

Understanding Media Queries

These Media queries use the CSS @media rule to conditions usually concern screen sizes and other device features. apply certain styles when specified conditions are met.

Basic Syntax

```
(condition) { @media mediaType and
unless conditions are met */ Can't style stuff
}
```

- **are:** Some of the common media types
- **screen:** For the computer screen, tablets, and mobile phones
- **print:**For printed pages and print previews
- **all:** For all media types (default if not specified)

Common Breakpoints

Breakpoints should be about content and not specific devices, but here are commonly used ranges:

```
/* Small devices (phones) */
(max-width:576px) {  @media
your kind words.  Thank you so much for
}
**/  /* Medium devices (tablets)
screen and (min-width: 577px) and (max-width: 768px) {  @media
/* Styles for tablets */
}
Large devices (laptops/desktops) */  @media (min-width: 1200px) {
/*
(min-width: 769px) and (max-width: 1024px){  @media screen and
/* Styles for laptops */
}
{ /* Extra large devices (large desktops) */  @media screen and (min-
width: 1200px)
screen and (min-width: 1024px) {  @media
```

For more details on how this works, read the first article of the series– paragraph styling in CSS.

```
 }
```

Mobile-First Approach

A mobile-first approach means to build for mobile devices first and then progressively enhance for larger screens:

```
 Inline styles */  Mobile base styles /*
.container {
width: 100%;
padding: 15px;
}
/* Adjust for tablets */
screen and (min-width: 768px){  @media
.container {
width: 750px;
margin: 0 auto;
}
}
/* Adjust for desktops */
```

```
screen and (min-width: 1024px){  @media
.container {
width: 970px;
}
}
adjust for large screens */}  {/*
and (min-width: 1200px) {  @media screen
.container {
width: 1170px;
}
}
```

Responsive Layouts

Flexible Grid Systems

```
.grid-container {
display: grid;
gap: 20px;
column on mobile */  grid-template-columns: repeat(1, 1fr); /* One
}
768px) {  @media screen and (min-width:
.grid-container {
repeat(2, 1fr); /* Tablet: Two columns */  grid-template-columns:
}
}
1024px) {  @media screen and (min-width:
.grid-container {
repeat(3, 1fr); /* 3 cols for desktop */  grid-template-columns:
}
}
screen and (min-width: 1200px){  @media
.grid-container {
minmax(0, 1fr)); /* Four columns on large screen */ grid-template-
columns: repeat(4,
}
}
```

Flexbox Responsive Patterns

```
.flex-container {
display: flex;
/* Stack vertically on mobile layout */  flex-direction: column;
```

```
gap: 20px;
}
{  @media screen and (min-width: 768px)
.flex-container {
are laid out horizontally on wider screens */ flex-direction: row; /*
Where the flex items
flex-wrap: wrap;
}
.flex-item {
2 items per row */  flex: 0 0 calc(50% - 20px); /*
}
}
{  @media (min-width: 1024px)
.flex-item {
calc(30vw - 20px); /* Three items per row */  flex: 0 0
}
}
```

Responsive Typography

```
:root {
  /* Base font sizes */
  --base-font-size: 16px;
  --h1-size: 1.75rem;
  --h2-size: 1.5rem;
  --body-size: 1rem;
}
body {
  font-size: var(--base-font-size);
}
h1 { font-size: var(--h1-size); }
h2 { font-size: var(--h2-size); }
p { font-size: var(--body-size); }
@media screen and (min-width: 768px) {
  :root {
    --base-font-size: 17px;
    --h1-size: 2rem;
    --h2-size: 1.75rem;
  }
}
```

```
@media screen and (min-width: 1024px) {
  :root {
    --base-font-size: 18px;
    --h1-size: 2.5rem;
    --h2-size: 2rem;
  }
}
```

Fluid Typography with Clamp

The clamp() function provides responsive typography without media queries:

```
h1 {
  /* Min size: 1.5rem, preferred: 5vw, max: 2.5rem */
  font-size: clamp(1.5rem, 5vw, 2.5rem);
}
p {
  font-size: clamp(1rem, 2vw, 1.25rem);
  line-height: clamp(1.5, calc(1.5 + 2 * ((100vw - 320px) / 1280)), 2);
}
```

Responsive Images

```
.responsive-image {
max-width: 100%;
height: auto;
}
*/
.art-direction {
*/  background-image: url('small. jpg'); /* Mobile default
}
768px) {  @media screen and (min-width:
.art-direction {
url(medium. jpg'); /* Tablet version */  background-image:
}
}
1024px) {  @media screen and (min-width:
.art-direction {
background-image: url('large. jpg'); /* Desktop version */
}
}
```

(HTML) with CSS styling Using the picture element

```
picture {
display: block;
width: 100%;
}
picture img {
width: 100%;
height: auto;
}
```

Advanced Media Query Features

Orientation

```
and (orientation: landscape) {  @media screen
portrait) {  @media(orientation:
.sidebar {
width: 100%;
height: auto;
}
}
(orientation: landscape) {  @media screen and
landscape) {  @media only screen and (orientation:
.sidebar {
width: 30%;
height: 100vh;
}
}
```

Display Quality

```
screen and (min-resolution: 192dpi){  @media
and (min-resolution: 2dppx) {  @media screen
and (-webkit-min-device-pixel-ratio: 2), only screen and (min-device-
pixel-ratio: 2) {  @media only screen
.hero-image {
url(‘hero-2x. jpg');  background-image:
}
}
```

using @supports Support Queries

When we combine media queries with feature queries, we can achieve some pretty awesome responsive design:

```
 screen and (min-width: 768px) {  @media
@supports (display: grid) {
.container {
display: grid;
fr));  grid-template-columns: repeat(auto-fill, minmax(250px, 1
}
}
not(display: grid) {  @supports
fallback since IE doesn’t support grid */  IE has to use a
.container {
display: flex;
flex-wrap: wrap;
}
.item {
width: calc(50% - 20px);
margin: 10px;
}
}
}
```

Reduced Motion

Media queries to address accessibility concerns:

```
 { }  @media (prefers-reduced-motion: no-preference)
.animated {
0.5s ease-in-out;  animation: fadeIn
}
}
{  @media (prefers-reduced-motion: reduce)
{animation: none;}  motion-reduce:reduce
.animated {
animation: none;
transition: none;
}
}
```

Dark Mode

```
:root {
--bg-color: #ffffff;
--text-color: #333333;
--accent-color: #3498db;
```

```
}
dark)  @media (prefers-color-scheme:
:root {
--bg-color: #1a1a1a;
--text-color: #f0f0f0;
--accent-color: #74b9ff;
}
}
body {
color: var(--text-color);
}
.button {
var(--accent-color)`  background-color:
}
```

Navigation Patterns Responsive

```
/* Mobile-first navigation */
.nav {
width: 100%;
}
.nav-menu {
appearance for mobile */  display: none; /* Default hidden
}
.nav-toggle {
display: block; ===> =  {menu} (for mobile> hamburger menu)
}
.nav-menu.active {
added  display: block; // show element when active class is
}
```

navigation Tablet and desktop

```
(min-width: 768px){  @media screen and
.nav-menu {
*/  display: flex; /* Visible, always on larger screens.
}
.nav-toggle {
none; // Bring back your hamburger menu */  display:
}
.nav-item {
margin-right: 20px;
```

```
}
}
```

Container Queries

Container queries, which are still gaining broader browser support, are progressive they are the future of responsive design:

```
 /* Define a container */
.card-container {
container-type: inline-size;
container-name: card;
}
*/  How it works:
{  @container card (min-width: 400px)
.card-title {
font-size: 1.5rem;
}
.card-layout {
display: flex;
}
}
card (max-width: 399px) {  @container
.card-title {
font-size: 1.2rem;
}
.card-layout {
display: block;
}
}
```

Design with Media Queries Responsive

- **Utilize relative units:** Use rem, em and percentage based values to have more adaptive layouts.
- **Content-based breakpoints :** Allow your content to dictate where the breakpoints should fall, not the particular dimensions of devices.
- **Avoid too many trackable points:** The golden rule is to have 3-4 major track points.
- **Test All the Time:** And not just resize a browser, but using real devices.

- **Think performance:** Keep the CSS in media queries to a minimum to avoid bloated code.
- **Use logical operators:** Use and, not, and only to chain conditions together for the most precise targeting you can achieve.

(max-width: 1024px) { @media screen and (min-width: 768px) and 768 < { @media screen and (min-width: 768px) and (max-width: 1024px)

/* Tablet-specific styles */

}

print */ Not Print[code] /* Everything except

@media not print {

screen { # 25. @media

}

Apply with a plan: Decide on mobile first or desktop first, then adapt it to your project. on the various devices with which they access the web today. With this knowledge, developers are able to create dynamic interfaces CSS media queries ensure the responsivity of a website, which is a key concept for designing comfortable experiences for web usershat respond thoughtfully to different viewing states of the application, all while balancing design and usability.

Multiple Choice Questions (MCQs)

1. **What does CSS stand for?**
   a) Computer Style Sheets
   b) Cascading Style Sheets
   c) Creative Styling System
   d) Custom Styling Script
2. **Which of the following is a correct way to apply an external CSS file to an HTML document?**
   a) <css href="styles.css">
   b) <link rel="stylesheet" type="text/css" href="styles.css">
   c) <style src="styles.css">
   d) <script href="styles.css">
3. **What is the default position value of an HTML element in CSS?**
   a) Fixed
   b) Absolute

c) Relative

d) Static

4. Which of the following is an example of a pseudo-class selector?

a) .container

b) #main

c) a:hover

d) div > p

5. **How can you make a div element 50% transparent using CSS?**

a) opacity: 50%;

b) opacity: 0.5;

c) transparent: 50%;

d) visibility: 50%;

6. What does the z-index property control?

a) Font size

b) Layer stacking order

c) Element width

d) Border thickness

7. Which CSS property is used to change the font of text?

a) text-style

b) font-family

c) font-style

d) text-family

8. How do you apply a background color of blue to a paragraph using CSS?

a) p { background: blue; }

b) p { bg-color: blue; }

c) p { bgcolor: blue; }

d) p { background-color: blue; }

9. **What is the purpose of the calc() function in CSS?**

a) Perform mathematical calculations for property values

b) Count the number of elements in a container

c) Convert CSS values to JavaScript functions

d) Apply transitions to elements

10. Which CSS unit is relative to the font size of the root element?

a) px

b) em

c) rem

d) %

Short Answer Questions

1. What are the different types of CSS?
2. Define CSS specificity and its importance.
3. How does the z-index property work?
4. What are pseudo-classes and pseudo-elements in CSS? Give examples.
5. Explain the difference between absolute, relative, and fixed positioning in CSS.
6. How can you center a div both vertically and horizontally?
7. What is the difference between em and rem units in CSS?
8. How can media queries be used for responsive web design?
9. What is the difference between opacity and visibility properties in CSS?
10. How do CSS animations work, and what are keyframes?

Long Answer Questions

1. Explain the different types of CSS with examples of their usage.
2. Discuss various CSS selectors, including element, class, ID, group, and pseudo-selectors.
3. Compare and contrast inline, internal, and external CSS. Which one is the best practice and why?
4. Describe the importance of the CSS box model and explain its components.
5. How do positioning properties (static, relative, absolute, and fixed) affect element placement? Provide examples.
6. Explain how CSS media queries work and give an example of a responsive design layout.
7. Discuss the role of CSS transitions and animations with examples.
8. How can CSS be used to create a gradient background? Explain the types of gradients.
9. What are flexbox and grid layouts in CSS? Compare their differences and use cases.
10. How can CSS be used to enhance user interaction by styling buttons and cursors? Provide examples.

MODULE 3
JAVASCRIPT

LEARNING OUTCOMES

By the end of this module, learners will be able to:

- Understand the basics of JavaScript, including data types and variables.
- Implement functions, loops, and control structures for scripting.
- Work with JavaScript objects and manipulate the Document Object Model (DOM).
- Handle user interactions through event handling and validate forms effectively.
- Explore ES6 features such as let, const, arrow functions, and promises.

Unit 7: Introduction to JavaScript and Its Fundamentals

3.1. Introduction to JavaScript – Basics, Data Types, and Variables

JavaScript (JS) is one of the most popular programming languages in the world, it acts as a scripting language for web development and is front-end-side part of web development along with HTML and CSS. JavaScript was initially created by Brendan Eich at Netscape in 1995 to make web pages more interactive as static pages. Now, it has matured into a multi-purpose language that drives everything from client-side web applications to server-side development using Node. js, mobile apps using frameworks such as React Native, and even desktop apps with Electron. Part of the widespread appeal of JavaScript comes from the fact that it is ubiquitous nearly every web browsers is shipped with a JavaScript engine that comes as standard equipment meaning it's the most available programming language on the planet. However, being a high-level, interpreted language with dynamic typing and first-class functions, might offer developers a great deal of flexibility in tackling problems (and thus also at times can create confusion for beginners who are used to robust typing). It's ECMAScript based, and ECMAScript 2015 (ES6) was a huge turning point in its development that had added lots of really powerful features that made certain prior complex programming patterns easier to code and the language itself became much more expressive and powerful. Before diving into JavaScript, it is essential to know the environment where it runs and executes. The JavaScript code is either directly embedded using the tag within an HTML document, or linked as an external file (with the. js extension. For page having JavaScript, using a web browser we can load that page. The browser processes page code by order by running code with the help of its JavaScript engine, for instance, chrome uses V8 engine, while Firefox uses a SpiderMonkey, and Safari uses JavaScriptCore (Nitro). Browsers have their own individual implementations that might differ a bit from one another, but they all follow the same ECMAScript spec, which means they behave consistently across platforms. By embedding JavaScript inside HTML files, the JavaScript code can also interact with the content of the page using the Document Object Model (DOM), enabling developers to build dynamic and interactive

user interfaces. JavaScript can run on the server with runtime environments such as Node. js, allowing for server-side applications and command-line tools. It is not surprising that their commonality is a major factor in the continuing success of JavaScript in modern software development as the glue language between different components of an application and the full-stack tool of choice. Before developers can write useful JavaScript programs, they need to understand JavaScript's fundamental? its basic syntax. The statements you write in JavaScript are written line by line and are normally followed by a semicolon (automatic semicolon insertion), which instructs the browser to execute the statements on each line one by one. It is case-sensitive, so user and User would be different variables. JavaScript enables developers to conveniently document their code and to disable sections of code without deleting them, as it supports single-line comments starting with // as well as multi-line comments that lie between /* and */. In JavaScript, the whitespace, which consists of spaces, tabs, and line breaks, is mostly ignored by the JavaScript engine (except for whitespace inside strings). As a result, whitespace can be used to format your code in a readable way without affecting the execution. This variability in syntax may make the language more accessible to new developers, but it means developers must develop coding conventions to ensure readability and maintainability of the code base, a necessity if multiple programmers share a code base since they may have different programming styles. JavaScript implements several primitive and complex types that developers are expected to know to write effective code, as data types form the base of any programming language. When you declare a variable with a value, the primitive types include Number (both integer and float), String (text in single or double quotes or template literals in backclean), Boolean (true or false), Undefined (a declared variable that doesn't contain a value), Null (an intentional absence of value), Symbol (a type introduced in ES6 to create unique identifiers), and BigInt (for representing a whole number larger than the Number type can safely accommodate). JavaScript doesn't differentiate between integers and floating-point numbers as some other programming languages do – any number is stored as a double-precision 64-bit floating-point value, resulting in errant behavior in some calculations that are looking for precise integers. JavaScript

strings are immutable, which means that once a string is created there is no way to modify its content, and because operations on strings always return new string representations instead of changing the existing value. Finally, booleans are necessary for controlling flows and executing conditional statements, and undefined and null, although both represent "nothing," serve different purposes for the design of the language.

JavaScript has complex data types in addition to the primitive types that enable you to write more complex data structures. The most base complex type is Object or a collection of key-value pairs, which is the building block for almost everything in JavaScript. And, one of the built-in Javascript objec that is specialized for ordered collections, is the array: such an object has numeric indices and a length property that automatically updates as you add or delete elements. In JavaScript, functions are first-class objects, so they can be assigned to variables, passed as arguments, and returned from other functions; this enables powerful programming paradigms like functional programming. RegExp Regular Expressions provide regex based searching for text. Date objects are created to represent temporal phenomena, but they have historically been criticized for the wackiness of their API, leading many developers to use external libraries, such as Moment. js, or the more recent native Intl APIs for the purpose of date manipulation. ES6 introduced Map and Set, providing more structured approaches to collection handling with unique keys or values, resolving the limitations of plain objects. So this is just a short summary of about your complex types in JavaScript being defined utilized in such a unique way that helps a user to understand the different methods and properties utilizing a very basic type. One such example is JavaScript which is dynamically typed that will allow you to declare a variable to hold values of any type and can reassigned values of any type during the execution of your program. This is in contrast to statically typed languages, such as Java or C++, where the data type of a variable must be declared, and cannot be changed. JavaScript has a type system based on type coercion——it automatically converts values from one type to another when operations involve mixed types and for example, adding a number to a string results in a concatenation of two strings, not an addition of two numbers. This behaviour must be properly understood

as it leads to subtle bugs at awothersihins in many situations, while being a huge time saver otherwise. In order to find out what type a value is, the typeof operator can be used, which returns a string that indicates the type of the operand. For primitive values, this usually gives the expected output (for example, "number", "string", "boolean"), but with a few quirks — most famously, typeof null returns "object" because of a historical bug that we keep for the sake of backward compatibility. For complex values, typeof generally returns "object", except for functions, where it returns "function". To differentiate between various kinds of objects more accurately, developers commonly use functions such as Array. isArray() or the instanceof operator that tests to see if an object has a given constructor somewhere in its prototype chain. In JavaScript, variables are named containers for values, and the language allows you to declare them using different syntax, each of which exhibits different behaviors. Unlike the var keyword, which was used before ES6, it has a function scope and declares variables global when declared outside of a function. But var has been largely supplanted by the more predictable let and const keywords (also introduced in ES6) that offer you block scope. let and const are ES6 features that allow you to define blocks, where let is mutable and const is that which is constant, but does not make its properties frozen, so an object assigned to a const variable can have its properties modified, but the assignment cannot be changed after initialization, so understanding how immutability works in JavaScript. Variable names can include letters, digits, underscores and dollar signs, but cannot start with a digit; they are usually written in camelCase, with the first of a series of words lowercase and subsequent words capitalized. Variable names should describe what they store, but also be short enough such that their usage remains clear. Following these conventions make the code maintainable and avoid name collisions in large codebases being worked on by multiple developers at the same time.

Have you ever thought variable scope in JavaScript? Variable scope in JavaScript is a crucial JavaScript concept to learn when you want to write effective JavaScript code and avoid unexpected behavior. The scope defines the accessibility of variables in the program and JavaScript has multiple types of scopes. The global scope covers the whole program; this means that a variable defined at this level can be

accessed from anywhere – that being said, over usage of global variables can create naming conflicts, and the code becomes less maintainable. Block scope of variables, which applies to variables declared via var within a function, restrict access to those variables to inside that function as well as any nested functions. Block scope (slightly more complicated) was introduced in ES6, through let and const to limit the visibility of variables to the block in which they are declared (curly braces), a scope like if statements, loops, and stand-alone blocks. Lexical scope (or static scope) means that where you can access a variable is determined based on the physical structure of the code, not the context of its execution at runtime – which is what enables closures, one of JavaScript's most powerful features, where functions retain access to variables of their containing scope after that scope has run. By understanding these scope mechanisms, developers can anticipate how and where their variables will behave in their code, and avoid common pitfalls like variable hoisting, a javascript specific mechanic that moves declarations (but not intitializations) to the top of their containing scope at compilation phase. Operators in JavaScript help you perform different operations on the values which can be as simple as arithmetic to as advanced as logical> Arithmetic operators are +, -, , /, % (remainder), * (exponentiation), and, for conversion to a number and changing sign, also rarely used + (unary plus) and — (unary negus). Assignment operators perform an operation and assignment in one concise step (+=, -=, *=, /=) to make it easier on the user. Comparison operators are used to evaluate the relationships between values and return boolean results: equality operators are both loose equality (==), which coerces types, and strict equality (===), which checks that types match, while relational operators are greater than (>), less than (=), and less than or equal to (<=). Logical operators) &&, ^^,!) – combines (or inverts) boolean expressions, but in JavaScript they return one of the operands, not necessarily a boolean value, so they allow some tricks like short-circuit evaluation and the nullish coalescing operator (??)). Bitwise operators work with the binary representation of values, whereas the ternary conditional operator (condition? expr1 : expr2) provides a shorter of conditional expression. String operators include: {#1} concatenation with the plus sign (+) and, added in ES6, template literals with backticks for more flexible string composition with

embedded expressions. Since then, as JavaScript has evolved, more data types, operators and features were introduced with each iteration of the ECMAScript standard to provide greater capabilities. ES6 (ECMAScript 2015) introduced template literals, enabling multi-line strings and string interpolation with ${expression} syntax, offering a more natural and readable way to compose strings. It also added the spread operator (...) which can unwrap iterables - like arrays - into individual elements for use in things like passing arguments to a function and manipulating arrays. Function declarations can also have default parameters, giving a fallback value when no arguments are provided. Destructuring assignment allows unpacking values from arrays or properties from objects into distinct variables with concise syntax. The optional chaining operator (?.) allows safely accessing nested properties without explicitly checking for each level's existence, preventing the infamous "cannot read property 'x' of undefined" errors. Newer additions like nullish coalescing (??) provide more predictable alternatives to logical OR (||) when dealing with falsy values that might be valid in certain contexts. These modern features have collectively transformed JavaScript development, reducing boilerplate code and enabling more expressive, less error-prone programming patterns that better reflect developer intent while maintaining compatibility with the language's foundational principles.

Unit 8: Functions, Control Structures, and DOM Manipulation

3.2. JavaScript Scripting – Functions, Loops, and Control Structures

One of the most important concepts in JavaScript programming is the concept of functions, which are reusable pieces of code that perform specific tasks when invoked. Traditional function declarations use the function keyword followed by a name, a parameter list in parentheses, and a code block in curly braces (e.g., function greet(name) { return "Hello, " + name; }). These declarations are hoisted to the top of their enclosing scope, so they can be called even before they are present in the code. While function declarations create functions which can be used before their own definition (e.g. function greet (name) { return "Hello, " + name; } can be invoked before the actual statement in the code flow), function expressions look like function declarations but instead of declaring a function, it will assign an anonymous or named function to a variable (e.g. const greet = function(name) { return "Hello, " + name; }), and also these function expressions will not get hoisted, so you can only use these functions after their declaration in the code flow. With ES6, function expressions can also be written using the new arrow functions syntax (e.g., const greet = name => "Hello, " + name;). In addition to providing unique syntax, arrow functions also show different behavior when it comes to this keyword, as they inherit it from their surrounding context, instead of creating their own, which gets rid of a typical error for standard functions. This behavior of binding context makes arrow functions especially useful in callback situations and when dealing with event listeners, where preserving the correct context can otherwise require explicit binding with the help of methods like bind(), call() or apply().JavaScript has several ways to pass values into functions, thanks to the function parameters. In the procedure declaration, you list the basic parameters, which are matched by position to arguments in the procedure call. In ES6, default parameters... meaning that the fallback values that are used when an argument isn't provided or is explicitly set to undefined (e.g. function greet(name = "Guest") {... }) Rest parameters are represented by the spread operator preceding the parameter name (such as function sum(...numbers) {... }), and they

enable you to gather multiple numbers into a single array within the function, allowing an arbitrary number of arguments to be taken by a function, which in contrast gives you a more structured way of working with arguments, compared to the arguments object, an array-like but not true array object with all parameters passed into a function. It can also be used when calling functions, where the spread operator expands iterables into individual arguments (e.g., max(...array)), leading to cleaner code in many cases. Another ES6 feature is parameter destructuring, which enables functions to extract values from arrays or properties from objects directly within the parameter list, simplifying the manipulation of complex data structures provided as arguments. Q: The how to get as much power as possible from JavaScript functions? A: Use parameters and return. It would work in other way as well, this is the trick you can use when using other if, use this interaction, returns and slots will help.Return values are the output that a function sends, serving as a means for a function to convey results back to where they were summoned. By default, in JavaScript, a function will return nothing (or, it will return undefined) unless there is a return statement, which is the case when the function has an explicit return statement or a return statement without return value. The moment a return statement containing a value gets executed, the function stops executing and the value is passed back to the caller. Functions can return any JavaScript data type, whether primitives, objects, arrays, even additional functions — this last one is what makes a higher-order function possible, one of the cornerstones of functional programming in JavaScript. You can return multiple values by packing them into an array or an object (an array is most commonly used to unpack the return — as the syntax for destructuring is more concise). [[functions can return other functions]] Functions can return other functions. This way, you will define patterns like closures, where an inner function still has access to a variable of its outer function after it has finished executing. This closure mechanism allows for data encapsulation, private variables, and function factories, where functions produce tailored functions based on their input parameters. Return values and how to deal with them effectively is something you must know to design functions that play nicely in your programs as a whole and also obeys DRY through separation of concerns.In JavaScript, functions have very different

variable scopes which have consequential effects on structuring code. Local variables defined inside a function using let, const, or var can be accessed only within that function itself, thus avoiding collision of names with variables from another scope, and allowing encapsulation of data.

The global scope, global variables are accessible throughout the program(global variables are declared outside of any function at top level & using var non-strict mode), so generally should be minimized. Closures are one of the most powerful features of JavaScript when a function continues referencing the variables inside its outer (enclosing) function scope even after the outer function has finished executing. Such behavior allows for elaborate patterns such as data privacy, where inner functions can read and write variables that are not visible to the outside world, effectively creating "private" state. Closures are also used to emulate private variables in JavaScript with the module pattern, a common design pattern in pre-ES6 JavaScript which allows us to use closures to create encapsulated code modules with private and public members. With the introduction of real modules in ES6, this avenue is less common of late, but the difference between close implementation and a simple function is important to recognize, and closures are fundamental in understanding the functional aspect of JavaScript and their occurrence in asynchronous programming and callback-style functions and event handlers.Control structures are statements that control the flow of a program's execution. Conditional Statements: It allows the codes to make decisions based on some condition. The if statement executes the block of code associated with the statement only when the specified condition evaluates to true, and an optional else clause is used to execute the block of code associated with an alternate execution path when the specified condition is false. Multiple conditions can be tested with else if clauses, which creates a chain of conditional tests. For scenarios that have many discrete cases, the switch statement provides syntax with clearer structure, evaluating the expression to execute the code block in the case with a matching case label. The ternary conditional operator (condition? If-else statements are replaced with the conditional operator as below (expr1? expr2 : expr3) It provides a way to do inline checks and return a value based on the condition. JavaScript has automatic type coercion behavior that can

lead to unexpected results when testing conditions, especially when using loose equality (==) — this is why the use of strict equality (===) is generally recommended so that constants of different types are not equal. Logical operators can be used anywhere to combine conditions (&&, ||) or negate them (!)., you only check the second operand when it's value is relevant based on the value of the first operand, making it well suited for concise conditional patterns. Knowing these conditional structures and their subtleties are basic to executing the decision making logic that is at the heart of nearly all non-trivial JavaScript applications.While control flow statements allow you to determine whether certain statements should run, loops go one step further; they enable you to execute code multiple times, which is critical when dealing with a series of data or needing to repeat a method a specific number of times. JavaScript's most general form of iteration construct is the most popular for loop, which comprises an initializer, a condition, and an increment expression, whose body executes while the condition evaluates to true. Every time the while loop runs, it checks the while conditions before running the next iteration. The do-while loop, which works similarly, guarantees the body executes at least once by checking the condition after the completion of each iteration instead of before. For eigenvalues, roll coins, the for...in loop iterates over all enumerable string properties including those inherited along the prototype chain – this behavior can sometimes be undesirable and care needs to be taken and additional checks in with hasOwnProperty() used to ignore inherited values. For iterating through arrays and other iterable objects, a for is better...for loop introduced in ES6, that iterates over the values of iterable, not the names of its properties. The break statement breaks entirely out of a loop, whereas a continue statement skips the current iteration of the loop and continues on with the next iteration. While they are not technically loops, array methods such as forEach(), map(), filter(), and reduce() allow us to get away from mutating data and state and use a functional approach to working on collections, aiding in code readability, and typically fewer lines of code. Note that deciding when to use one loop or another is very important in order to write more efficient and readable JavaScript code that can easily handle different types of iterations needs.If you're manipulating collections, for example, built in array methods are a very powerful

alternative to loops, leading to more declarative styles of programming.

Another common approach is using Array.prototype.forEach() as a simpler syntax than the explicit loops, but with no way out from, for example, premature termination of iteration: The map() method creates a new array with the results of calling a provided function on every element in the calling array, excellent for manipulating all elements in an uniform way. The filter() method returns a new array, containing all elements of the calling array for which the provided filtering function returns true, perfect for creating subsets based on a given criteria. The array reduce() method executes a reducer function on each element of the array, resulting in a single output value, allowing for complex aggregation and transformation. The find() method returns the first element that fulfills a provided testing function, while findIndex() returns the index of that element instead. As the names imply, the some() and every() methods test whether some or all elements pass a given test respectively and will return boolean results. These higher-order functions promote thinking about operations at a higher level of abstraction, thinking about what might be done as opposed to how it might be iterated on to accomplish a goal, and often leads to shorter, more readable, and less error-prone code with respect to common looping errors such as of-by-one or a broken exit clause. The try-catch statement is the cornerstone of error handling that lets code try actions that can fail (the try part) and handle these errors gracefully (the catch part). You can define an optional finally block that is executed whether or not an error occurred, which is useful for cleanup operations which need to always execute. By subclassing the Error class or its more specific variants, such as SyntaxError, ReferenceError or TypeError, the developers can implement custom errors and they are able to describe exactly what went wrong in the context of their application. The throw statement programmatically throws an error; it takes an arbitrary value as the error object, but by convention, we throw an instance of Error or that of its subclasses, since it will capture where it comes from by tracking the stack traces. Error handling is more complex in async code — promises can get.catch. catch() methods or the second argument to. then() which offers an alternative to error handling, async/await designates the option of using more traditional try-catch blocks with

async operations, thus combining more familiar error handling syntax. Error handling helps to catch exceptions and handle errors gracefully, preventing crashes from the application when unexpected situations occur, it also helps us in debugging our application by showing us meaningful error messages and provides the user experience by showing the appropriate messages when things don't go as expected and operations can't be performed.This article covers how the execution model in JavaScript affects how the code works, especially in relation to timing and asynchronous operations. The one live outer event loop handles the language on a single thread, where one operation at a time is processed from the call stack, which keeps track of the currently running functions. It also raises the possibility of making blocking calls that can make the user interface become unresponsive for a long time. As a solution to this problem, JavaScript supports asynchronous programming via several methods. The traditional approach is called Callbacks which consists of functions passed as arguments that are executed once some operations are complete. Promises, which were introduced in ES6, stand for a value that might not be available yet and have methods such as. then() and. catch() for successful and failed operations respectively, and even allowing more structured chains of asynchronous operations. Async/await allows us to write asynchronous code in more of a "synchronous" style which can be a very nice change of pace from the callback hell we can encounter with Promises. Web APIs such as setTimeout(), setInterval(), and fetch() work outside of the main thread, putting their callbacks in the task queue once they finish, which the event loop moves to the call stack once empty. Making sense of this execution model is the key to building responsive web applications that can execute resource-intensive operations or network requests without freezing the UI, In modern web development, where UI responsiveness directly contributes to user experience, this is an important consideration.The fact that modules are needed in JavaScript illustrates the history of namespace pollution in global scope, and modules are a way to separate units of code that can be reused. Before ES6, developers used patterns such as Immediately Invoked Function Expressions (IIFEs) as well as tools like RequireJS or Browserify to mimic the functionality of modules. With the import and export keywords, ES6 introduced module support natively in

JavaScript, making it possible for individual JavaScript files to explicitly declare what they provide to other modules, and what they need from other modules. Named exports (export function, export class, export const) allow for multiple exports from a module, whereas default exports (export default) enable a default export, which is ideal for modules that only export a predominant feature. Modules are automatically executed in strict mode, which helps catch common coding mistakes. They are also run just once on import, returning a reference to the same module instance on each import, so that state is consistent within an application. Unlike scripts, which run immediately when the browser gets to them, modules are deferred by default, meaning the browser won't run them until the document itself is fully parsed. Development environment tools current-generation bundlers and build systems, such as Webpack, Rollup, and Parcel now support ES modules, even offering tree-shaking to improve performance by omitting dead code from included modules. Modules are a major step forward in the maturation of JavaScript as a language for building larger applications, bringing better organization, reusability, and maintainability to complex projects.

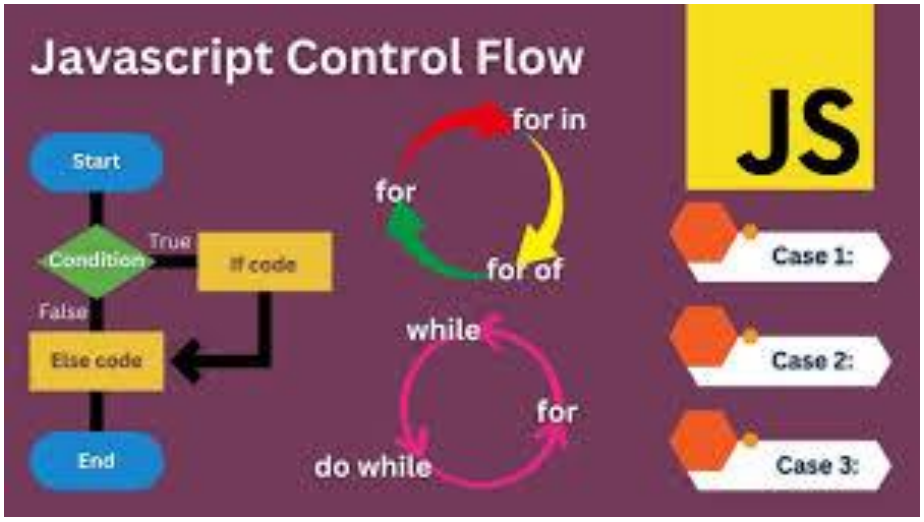

Figure 3.1: Javascript Control Flow

(Sorce: https://www.electroniclinic.com)

3.3. JavaScript Objects and DOM Manipulation

Objects are the building blocks of JavaScript as a way to store collections of related data and functionality. Simply put, an object is an unordered collection of key-value pairs, where keys (or property names) are strings or symbols, while values can be any JavaScript data-type — from primitives to other objects and functions. You create objects either using an object literal (i.e., an expression enclosed by curly braces e.g., {name: "John", age: 30}), Object constructor (new Object()) | constructor functions / classes that act like templates to create similar or similar structured objects. You can access properties through dot notation (object. You can access the properties in dot notation (object.property) or bracket notation (object["property"]) — bracket notation is needed when property names contain special characters, whitespace characters, or those dynamically determined at runtime. Regular objects have no order (sure, modern JavaScripts keep the insertion order for non integers) unlike arrays which are ordered. Assignment operators or the delete keyword can be used to dynamically add, modify or delete properties from an object at any point during the execution of a program, demonstrating the dynamic nature of JavaScript. The ability is powerful but can lead to subtle bugs in larger applications if properties get modified or deleted unexpectedly [when all property names are dynamic strings]. In summary, object properties have several aspects that define their behavior and visibility. Data properties hold values and are the most common type of property. Accessor properties, which are created with get and set methods, run code when properties are read or written, providing computed values and validation logic. Property Attributes - Object.getOwnPropertyDescriptor()}} The post Property Attributes appeared first on Object. getOwnPropertyDescriptor(), control behavior of properties : configurable specifies whether property can be deleted or change its attributes enumerable controls property visibility in such kind of iterations as for...in loops; writable determines whether the value of a property is changeable and value is the actual data for data properties. Object can be used to configure these attributes. defineProperty() or Object. defineProperties() with fine granularity of control about object behavior. Property descriptors are a way to work with these attributes programmatically, and

outsource some advanced patterns: data validation, computed properties, property protection. Since ES6, objects can have symbols as property keys, allowing properties to be created that do not participate in normal iteration methods, facilitating the addition of "hidden" properties or methods that won't conflict with existing or future property names, a technique used by many built-in JavaScript methods and frameworks to store internal state on objects without the risk of collision with application-defined properties. Prototypes are the basis for JS inheritance, enabling objects to inherit properties and methods from other objects. In JavaScript, each object has an internal link ([[Prototype]] in ES5] to another object which is its prototype, and it is this that it inherits properties from when they don't exist on the object. When property is accessed on any object JavaScript first checks the property on that object and then it on its prototype chain until it finds some object with null prototype which is object generally. prototype. This chain is the basis of prototype-based inheritance, in contrast to class-based inheritance in Java, C++, and other languages. When a constructor function is invoked with the new keyword, it creates an object whose prototype is set to the constructor's prototype property. The Object. The create() method provides a straightforward approach to establishing prototype links between objects without using constructors. Classes: There is a class syntax added in ES6 too, and it is syntactic sugar over the prototype-based inheritance of JavaScript rather than an entirely new inheritance model, but for developers coming from class-based languages it provides a more familiar syntax. The instanceof operator determines if an object has a specific constructor in its prototype chain, and Object. getPrototypeOf() is an object.[[Prototype]] accessor. Prototypes are key to learning JavaScript's object-oriented features and to gaining understanding of how inheritance works in built-in objects and third-party libraries.

Constructor Functions : They Function as Templates for Creating Objects With Uniform Structures and Behaviors. A classical constructor is just a function that is designed to be called with the new keyword, which creates a new object that inherits from the constructor's prototype property. It is a convention that constructors should start with a capital letter to distinguish them from functions. The this keyword is a keyword most commonly used inside a

constructor, which refers to the object being created and assigns properties directly (this. name = name). Because methods are generally defined on the constructor's prototype (as opposed to preventing the same thing being defined in many different instances), they only exist in one place in memory, no matter how many instances exist. The class syntax was introduced in ES6, which gives developers a more familiar and concise way of defining constructor functions and their prototype methods. However, JavaScript's class syntax is still prototype-based implementation under the hood which should be familiar to whoever comes from a class-based language. Classes pass constructor methods for initialization, instance methods which show up on the prototype, static methods which attach to the class rather than an instance, and getter and setter methods for computed properties, and extends syntax for sub-classing. Since then, orthodox Javascript has adopted Red and many of the experimental syntaxes in a much more readable form, making working with object oriented Javascript code much more maintainable, but familiarising oneself with the underlying prototype mechanism is still useful for debugging and working with older codebases. The Document Object Model (DOM) is a programming interface for HTML and XML documents, and it represents the structure of a web page as a tree of objects, allowing JavaScript to interact with and manipulate the content of the page. The document object, which represents the entire HTML document and serves as the entry point for DOM manipulation, is the root of this tree structure. According to the previous definition, nodes are the building blocks of the DOM tree and there are several node types: element nodes (representing HTML elements), text nodes (representing text between the elements), attribute nodes (representing element attributes) and comment nodes (representing HTML comments). The hierarchy reflects the nesting of HTML elements, with the parent-child relationships between nodes corresponding to the containment relationships in the original markup. This hierarchical structure supports traversal in all directions: parent to child, child to parent, and between siblings. There are several properties that help you accomplish this task: parentNode, childNodes, firstChild, lastChild, nextSibling and previousSibling. The Document Object Model enables JavaScript to identify parts of a web page and makes it possible for JavaScript to change, add, or

remove portions of the web page in response to user or external events, why you need to have a basic understanding of the DOM structure in order to effectively work with web technologies. It can be used to find, change, add or remove elements in a web page using javascript code. The first step in this process is about selecting elements, using methods like getElementById(), getElementsByClassName(), getElementsByTagName() and the more flexible (and powerful) querySelector() and querySelectorAll(), which use CSS selector syntax to identify elements. And the various ways that you can change those selected elementsOnce you've selected something, you can change it in a million different ways: innerHTML changes the HTML inside an element, textContent inside an element text, style gives you access to inline CSS properties, classList lets you add, remove and toggle css classes, setAttribute() and getAttribute() let you manipulate element attributes. New elements are created by calling document. createElement() method, use appendChild() or insertBefore() or other methods to insert them at specific position in the document. You need to select the element to be removed and call remove on the element, or removeChild on the parent element. Using cloneNode() you can clone elements either shallow or deep, modern DOM manipulation techniques are typically facilitated through higher-level abstractions provided by libraries (jQuery) or frameworks (React, Angular, Vue) that address the complexity of efficient DOM updates, cross-browser compatibility, and synchronization of application state with visual UI elements that make web development for complex interactive applications much easier.

Events are a way for JavaScript to react to user actions and other activities that happen in the browser environment, they produce the core of interactive web applications. When events occur it means that we have an event-driven programming model and that means that JavaScript waits for an Event and acts on it. These events can include click, mouseover, keydown, submit, load, resize, and more, which all indicate different interactions or browser actions. The functions that get executed when an event occurs are called event handlers, which can be attached to the elements in multiple way: inline HTML attributes (eg: onclick="handleClick()"), element properties (eg: element. onclick = handleClick), and the more flexible addEventListener() method, which allows multiple handlers for the

same event type on one element. Keep in mind that event handlers automatically pass in an event parameter, which is an object that contains information about the event: for example, its type, the element or elements involved, and event-specific data like mouse coordinates or key codes. Event propagation occurs during two phases: the capture phase, which traverses downwards from the document root to the target element; and the bubbling phase, which traverses upwards from the target back up to the root. The stopPropagation() method stops this propagation while preventDefault() prevents the default action from the browser for the event like opening a link or submitting a form. Custom events with the CustomEvent constructor allow decoupled communication between application components. If you want to write web app which is responsible to user interaction and responsive to browser environment changes you have to well- versed with the event handling. Forms are an important part of web applications, they help collect data from users to be processed on the client or submitted to back-end servers. If you want to access a form from js, you can do this through the DOM, usually by selecting the form itself using getElementById() or querySelector(), and then access its elements using the elements collection or directly selecting specific inputs. The submit event is fired when the form is submitted; it allows you to validate input before sending the data to the server. PreventDefault(): Event handlers for this event usually make use of preventDefault() to prevent the submission from happening if the validation fails or if using AJAX-based submission techniques. HTML5 attributes such as required, pattern, or min/max and JavaScript logic checking input values against any given criteria can be used for form validation. Thus we can give real-time validation feedback as we are typing in the form fields through input event changes such as change, input, and focus events. Once validation has been performed, the data can be collected from a form in a number of ways: directed at the input value properties, using the FormData API which offers a more structured representation, or by using the newer form:. elements approaches. 91 // You can submit forms programmatically by calling the form's submit() method: 92 form.submit() 93 // or use the Fetch API or XMLHttpRequest to send the form data asynchronously, 94 // allowing more fluid user experiences where pages do not need to

reload upon form submission. AJAX (Asynchronous JavaScript and XML) allows web applications to communicate with servers and update portions of a web page without the need to refresh the entire page, which greatly enhances the user experience, making web applications feel more like desktop applications. Ajax calls were primarily done using the XMLHttpRequest object historically which we pass in the methods to open the connection, send the request and get the response using callbacks and event listeners. Modern JavaScript applications now almost exclusively use Fetch API built as a more capable and flexible replacement for it and returns promises that simplify writing and maintaining asynchronous code. Fetch takes a URL and an optional options object for setting the request method, headers, the body content, and other settings. As such, response handling will start with checking the response status, followed by checking the body in the appropriate format – most often JSON using the. json() method, but in addition as textual content, blob, or different codecs relying on the content-type. Due to the fact that fetch only rejects promises for failures on the network level, not http-level errors like 404 or 500, we are required to check for errors explicitly. Security must be respected when working with CORS guidelines, which order browsers (the developers of the same) to not send requests to domains not serving the page unless the appropriate headers allow that. For more complex needs, libraries like Axios build on these native capabilities with features such as auto JSON parsing, request/response interceptors, and streamlined error handling. Web Storage APIs enable web apps to store data on the client so that they can maintain state across page refreshes and browser sessions. The LocalStorage on the other hand provides a place to persist data across sessions, which will be available until the user intentionally clears it or it is explicitly removed by the program – great for preferences, settings or cached data with long term lifetime. SessionStorage offers temporary storage for as long as the current browser tab is open, so if the tab or window is closed, the stored data will be lost; use this for retaining data state between user interactions in the same visit, like a multi-step form or wizard, These two types of storage implement the same key-value API, featuring methods like: setItem(key, value), getItem(key), removeItem(key), clear(). Although it can store key-value pairs, all values will be strings and complex data types will

require serialization/deserialization (usually using a JSON for this purpose). stringify() and JSON. parse(). Storage capacity varies by browser, but usually round to 5MB to 10MB, with a clear maximum that is a big advantage over cookies, which are capped at approximately 4KB. Unlike cookies, the storage data is not sent along with every request to the server, making it a good choice for applications that store a lot of data on the client-side and therefore will add a lot of data to outgoing requests. Storage events allow cross-tab/window communication to listen for storage value changes, which is helpful when multiple instances of the same app are opened simultaneously.

3.4. Event Handling and Form Validation

Events like mouse clicks, keystrokes, form submissions, window One some sort of web development knows how event propagates or how to validate a form properly. web applications is event handling, which allows developers to build interfaces that respond to user actions. This anyone who has been working with The key behind interactive bubble up through its ancestors in the DOM tree. resizes, and so on. By default the JavaScript event model uses a bubbling pattern, meaning that events trigger from the target element at source then of the great core features offered by the Document Object Model (DOM) is a set of structures that represent all elements of a web page along with associated events.

```
const button = document. getElementById('myButton');
function (event) {  button. addEventListener(click,
console. log('Button was clicked!' );
//2976.0861/ }}{{{{2670. api apimakeup /\.( (a. preventdefault. event. addEventListener());
});
```

This Event delegation, which uses event method is especially beneficial for dynamically generated elements or when working with extensive lists. bubbling to manage events for several elements with a single event listener, is a potent pattern.

```
// Event delegation example
function(event) { document. getElementById('parent-list'); addEventListener("click",
if (event. target. tagName === 'LI') {
item:', event. target. textContent);  console. log('Clicked on list
```

```
}
});
```

Form Validation: Form validation is a critical part of Forms are an essential part of web applications, user convenience to prevent any unwanted data submission and for us as a system to secure it through server-side validation. web development that ensures user-submitted data is valid and safe to process. Thus client-side validation is important for they are the main way for users to enter data.

These More recent validation additions like the features enable browser-to-browser validation with minimal JavaScript involvement. required, pattern, min, max, and type attributes (all introduced with HTML5) were in many ways the natural evolution of validation on the web.

example of how to get the email. $email = $_POST['email']; // Just as an

The Constraint Validation API extends the built-in validation mechanism, However, for more complex validation scenarios, you can always leverage JavaScript to create custom validation providing developers read access to validation state and custom error messages.

registrationForm = document. const form = document. const

plugin with the above parameters. const email = document. You are not allowed to use the

```
function() {  email. $('input').on('input',
if (email. validity. typeMismatch) {
email address');  email. setCustomValidity(Please specify a valid
} else {
email. setCustomValidity('');
}
});
) {  form. addEventListener( 'submit', function( event
if (! form. checkValidity()) {
event.preventDefault();
```

(part No. 3) Use Error handling in a basic and advice based simple way. Error Handling

```
showValidationErrors();
}
});
```

validateEmail: Validation rules are simply Extract string from more var with Regular Expressions Regular expression is very useful Putting it all together, we can define a function, expressions that can be evaluated as True or False. These rules are expressed using the Validator built-in functions. while validating any data like email address, phone number, postal code, etc.

const regex = /^[a-zA-Z0-9. _%+-]+@[a-zA-Z0-9.-]+\. [a-zA-Z]{2,}$/;

return regex.test(email);

}

As users can fix errors earlier in the input process, it leads to less frustration users can face while filling the forms, Using real-time/inline leading to higher completion rates. validation helps user experience as it gives feedback input-wise, i.e., as the users type instead of waiting until form submission.

name="password"]'); your favorite thing about it? querySelector('input[What's

document.getElementById('password-strength'); passwordStrength = document. password-strength = const

text entry. passwordInput. This event fires when the user inputs a value during

= calculatePasswordStrength(this. value); const strength

${strength}; passwordStrength. textContent = The password strength is:

= strength- + strength. toLowerCase()}`; passwordStrength. className

});

However, older browsers might need to fallback to another solution because sometimes, the Constraint Validation API might not be When we implement Yup would help to keep the same validation behaviour for you all the time. supported in older browsers, although newer versions of popular browsers would offer standard coverage. Libraries such as Validate. The solution you mentioned (using js, Formik, or form validation, we still need to take care of cross browser compatibility. Do show clear, descriptive error messages that Form validation is another key area in which accessibility comes a more accessible form validation experience. all users will understand

including those using screen readers. The aria-invalid and aria-describedby attributes can aid in providing into play.

{ showError(input, message)

= document. getElementById(${input. name}-error); const errorElement

errorElement. textContent = message;

multiple sclerosis, or rheumatoid arthritis. input. or topical use, you may be able to use this with diseases such as AIDS, cancer,

${input. name}-error); input. setAttribute('aria-describedby',

}

These frameworks handle a lot of complexity of You can also use the same type of form handling and validation with modern frameworks such form validation under the hood with ample customization capabilities for custom needs. as React, Angular, and Vue, all of which include a state management solution. It must include clear visual indicators, For validation failures the overall user experience. meaningful error messages, and the preservation of valid data in the event of an error. An effective error handling mechanism plays a significant role in should be properly handled through error strategies.

3.5. JavaScript ES6 Features – Let, Const, Arrow Functions, Promises

In contrast, these are some core features of modern JavaScript development, providing syntactic sugar, better scoping and powerful One of the most comprehensive revisions to JavaScript since its async programming capabilities. inception came in ECMAScript 2015 or ES6.

When you declare variables with the let keyword, these variables are block scoped; meaning they are scoped to the nearest enclosing block (enclosed by curly braces), resulting in more predictable code behavior with Ready fewer unintended side effects. for something small but really good.

```
// Block scoping with let
if (true) {
'I exit only in this block';  let blockScoped =
of the function';  var functionScoped = 'I am alive for the life
}
// Good  console. log(functionScoped);
defined console. log(blockScoped); // ReferenceError: blockScoped is not
```

While a variable is constant, the contents of that constant variable itself can still change; a The const keyword allows you to create variables that reuse a distinction that is critical in using objects and arrays. reference but do not let you reassign that reference, which is (effectively) how you declare constants in JavaScript.

```
const PI = 3.14159;
variable  PI = 3.14 // TypeError: Assignment to constant
'Alice' };  const person = { name:
contents of the object can be changed  person. name = 'Bob'; //pros -
// TypeError: Assignment to constant variable  const person = { name: 'Charlie' };
```

But, in addition about arrow function (= Now, before talking>method chains. to being shorter, arrow functions lexically bind their this value, so you never have to worry about that dynamic binding issue we always had with functions. This is particularly useful in callbacks

and), Arrow function is a special shorthand for writing a function expression.

```
// Traditional function
function add(a, b) {
return a + b;
}
// Arrow function equivalent
const add = (a, b) => a + b;
// Lexical 'this' binding
const counter = {
count: 0,
increment: function() {
function if you want to preserve 'this' context)  // (Use arrow
setInterval(() => {
this.count++;
console. log(this. count);
}, 1000);
}
};
```

This allows us to define functions very easily, and remove a lot of Default parameters are one of the available forms boilerplate code that we need to write for just parameter checking. of function parameters.

```
'Hello' ) {  function greet( name = 'Guest', greeting =
return ${greeting}, ${name}! ;
}
greet(); // "Hello, Guest!"
greet('Alice'); // =>"Hello, Alice!"
"Hi, Bob!"  greet('Bob', 'Hi') //
```

This Backticks (`) are used for template literals that allow string interpolation and multi-line strings without using makes it easier to read and maintain string manipulation code. concatenation or escape characters.

```
const name = 'Alice';
= `Hello, ${name}!  const greeting
Welcome to our website.`;
```

The output can then be used to get the value of the keys directly without further processing, which The destructured assignment syntax is a shorter way of extracting values from arrays or properties

is especially useful when dealing with nested data structures or API responses. from objects and assigning them to variables.

```
// Array destructuring
2, 3, 4, 5, ];  const[first, second,...rest] = [ 1,
// Object destructuring
'Developer' } = { name: 'Alice', age: 30 }; const { name, age, profession =
```

The Spread and rest operators (...) offer flexible methods for spread operator takes an iterable and expands it into its constituent elements, whereas the rest operator takes multiple elements and combines them into a single entity. working with arrays and objects.

```
// Spread operator
const arr1 = [1, 2, 3];
5]  const arr2 = [...arr1, 4, 5]; // [1, 2, 3, 4,
// Rest parameter
function sum(...numbers) {
+ num, 0);  return numbers. reduce((total, num) ⇒ total
}
```

Async/await offers a cleaner way to write asynchronous code by Promises are avoiding the "callback hell" problem that earlier JavaScript asynchronous code faced. a great improvement in the way JavaScript handles asynchronous operations. This successful, or failed. It A Promise is an object that tries to see if an async operation is finished, renders asynchronous code more predictable and easier to reason about. can be in one of three states: pending, fulfilled, or rejected.

```
// Creating a promise
const FetchData = new Promise((resolve, reject) =>{
// Asynchronous operation
setTimeout(() => {
const success = Math. random() > 0.5;
if (success) {
resolve({ data: 'Success!' } );
} else {
to fetch data')  reject new Error('Failed
}
}, 1000);
});
```

```
// Consuming a promise
fetchData
. then(result => console. log(result. data))
. catch(error => console. error(error. message));
```

All then methods return a new Promise, which allows for a flat structure of async Using promise chaining, you code. can write sequential asynchronous actions explicitly, without putting callbacks in nested levels.

```
fetchUserData(userId)
..then(userData =>fetchUserPosts(userData. postIds))
=  ..then(posts> posts.filter(filterRelevantPosts))
=  . then(relevantPosts> displayPosts(relevantPosts))
=  ..catch((error)> handleError(error));
```

One of the advantages of Promise.all() is that it allows you to handle multiple promises in parallel, before a subsequent step can occur. and wait for them all to resolve before continuing. This is especially helpful when different asynchronous operations need to complete The Promise.

```
const promises = [
fetchUserProfile(),
fetchUserPosts(),
fetchUserFriends()
];
Promise.all(promises)
. then(([profile, posts, friends]) =>{
// All data available here
friends: UserFriends) renderUserDashboard(profile: UserProfile, posts: UserPosts,
})
.catch(error => {
```

automatically resolves and if any, rejects catch is called. All the above promises

```
showErrorMessage(error);
});
```

It The Promise. Returns: a is useful for implementing timeouts or to arbitrarily select the fastest available resource. promise that resolves or rejects as soon as one of the promises in an iterable resolves or rejects, with the value from that promise.

```
fetchData();  const dataPromise =
=  const timeoutPromise = new Promise((resolve, reject)> {
=  setTimeout(()> reject(new Error('Request timed out')), 5000);
});
timeoutPromise])  Promise. race([dataPromise,
=  . then((data)> processData(data))
. error =>handleError(error)));
```

Even though JavaScript is still prototype-based under the covers, the class syntax provides a more The features of classes in ES6 allow you to create object-oriented code with a cleaner and more familiar syntax and deal with traditional way for developers formerly of class-based languages to inertia object-oriented programming patterns into JavaScript. inheritance in a more intuitive way.

```
 class Person {
constructor(name, age) {
this.name = name;
this.age = age;
}
greet() {
Hello, my name is ${this. name};  return
}
static createAnonymous() {
Person('Somebody', 0);  return new
}
}
person {  class Employee extends
) {  constructor( name, age, position
super(name, age);
this. position = position;
}
greet() {
${this. position};  return ${super. greet()} and I am a
}
}
```

The use of import and export statements enforces a clearer understanding of dependencies and ES6 introduced modules, a clean avoids polluting the global namespace. way to share code across JavaScript files.

```
// math.js
export const PI = 3.14159;
export function square(x) {
return x * x;
}
// app.js
import { PI, square } from '. /math. js';
import * as math from '. /math. js';
```

With the introduction of block-scoped variables, arrow functions with lexical this binding, and promises for asynchronous programming, JavaScript applications were fundamentally All of these ES6 features together mark a huge step forward for JavaScript, paving the structured and developed in a completely new way in the professional web landscape way for developers to produce neater, more maintainable and more functional code.

3.6 Introduction to AJAX and JSON

The concept changed the way we do things from a request- response to interactive Asynchronous JavaScript and XML (AJAX) changed the game of web development by allowing web apps to revise content in response user experience which led to modern single-page app. to user input without needing to reload entire pages. These The essence of AJAX is sending asynchronous http request from the client-side desktop-like experience. requests occur behind the scenes, enabling users to keep reading a page while data is retrieved or sent. The JavaScript code receives the response from the server, and updates appropriate pieces of the page with the new information, creating a more responsive, javascript codes to the server.

Simple AJAX request (XMLHttpRequest) //

```
xhr = new XMLHttpRequest(); var
true); xhr. open('GET', 'https://api.example.com/data',
xhr. onreadystatechange=function(){
if (xhr. readyState === 4 && xhr. status === 200) {
const response = JSON. parse(xhr. responseText);
updateUIWithData(response);
}
};
xhr.send();
```

Though still supported by all browsers, it has been largely replaced by the much more modern The original implementation of AJAX was held together with glue and string around the XMLHttpRequest object, for opening connections, Fetch API, which exposes a cleaner Promise based interface. sending requests and receiving responses.

Modern AJAX request using Fetch API some data from the server without a page refresh. //

-- After running above code you can call it through /api/data> properties.

```
.then(response => {
if (!response.ok) {
throw new Error(HTTP error! Status: ${response. status});
}
return response.json();
})
. then(data => with data)  update UI
. catch(error => error);  console. error('Fetch error:',
```

JSON data is structured as key-value pairs, using a syntax similar to JavaScript objects, making JSON (JavaScript Object Notation) was JSON, the de facto standard format for AJAX applications data exchange, though its it intuitive for developers to work with. simplicity, lightweight, and direct JavaScript parsing.

```
{
"id": 123,
"name": "John Doe",
"email": "john@example.com",
"roles": ["user", "admin"],
"settings": {
"notifications": true,
"theme": "dark"
}
}
```

a json string. These are used to work with the JSON data returned by an method parses a JSON string and returns a JavaScript object and JSON. - stringify(): it converts a js object to The JSON. parse() AJAX application.

JavaScript objects interaction — converting JSON to

```
const user = {
```

name: 'Alice',
age: 30,
draw lots of hobbies into that list as possible. You
};
= JSON. const jsonString = JSON. var jsonString
// Back to JavaScript object const parsedUser = JSON. parse(jsonString);
REST is an architectural style consisting of constraints RESTful for designing web services, focusing on stateless client-server communication, cacheable responses, and uniform interfaces.11 APIs (which stands for Representational State Transfer) have emerged as the most significant architectural style for web services, offering a widely-adopted specification for structuring AJAX interactions.
RESTful API CRUD using Fetch //
// GET request (Read)
get('https://api.example.com/users/123')
. then(response => response. json())
= . Then(user> displayUser(user));
// POST request (Create)
{ fetch('https://api.example.com/users',
method: 'POST',
headers: {
'application/json' 'Content-Type':
},
'alice@example.com'}) body: JSON. stringify({ name: 'Alice', email:
})
. then(response => response. json())
. then(response =>addUserToList(response.data));
To domains. Similar-origin policy Cross-Origin Resources Sharing (CORS) is a browser security feature that restricts what AJAX requests can be made to different allow a web resource to be called from a different domain, the servers must enable CORS by providing these headers in their response. The same-origin policy is designed to prevent potentially malicious scripts from accessing data across domains.
CORS example) Requesting data from another domain (remember
{ fetch('https://otherdomain.com/api/data',
request credentials: 'include' // Send cookies with the

```
})
. then(response => response. json())
=  ..then(data> processData(data))
.catch(error => {
instanceof TypeError) {  if (error
console.
}
});
```

a demand does not resolve. Both XMLHttpRequest and Fetch also offer ways to handle errors Hence, dealing with error in AJAX applications is something significant, if you need to inform the user to guarantee great user experience when of different kinds, such as with the network itself, or server code.

```
 function fetchData(url) {
return fetch(url)
.then(response => {
if (!response.ok) {
an error status code  // Server replied with
Error(Server error: ${response status} ${response. statusText});
throw new
}
return response. json();
})
.catch(error => {
parsing error  Error: Network error or JSON
console. error('Fetch error:', error);
error. Please try again later.' );  showErrorMessage('Data loading
storage data (optional)  // Retry the request with backoff or use
return getCachedData();
});
}
```

Since things like requests happen asynchronously, it is important to update users regarding data being loaded or processed so that they are aware of what is happening in the application and to avoid their confusion regarding Despite their efficacy, AJAX applications can state. be datatious in themselves, because they are built to respond when a action has been executed.

```
function loadUserData() {
```

```
showLoadingIndicator();
fetch('/api/user-data')
. then(response => response. json())
.then(data => {
updateUserInterface(data);
hideLoadingIndicator();
})
.catch(error => {
data');  showErrorMessage('Something went wrong in loading
hideLoadingIndicator();
});
}
```

It can help AJAX application with fetching data faster by allowing data once retrieved to access to the data that is not frequently modified. be stored locally for subsequent calls. This helps minimize unnecessary network requests and enables quicker Using Client-side cache to optimize AJAX application performance:

```
const cache = new Map();
{  function fetchWithCache(url, expirationTime = 60000)
const cachedData = cache. get(url);
const now = Date.now();
if (cachedData && cachedData. timestamp response. json())  now -
.then(data => {
now })  cache. set(url, { data, timestamp:
return data;
});
}
```

Regular polling You make declarations of requests periodically to see if there is some data Polling and long polling are used in AJAX available Long-polling Keep the connection open until data becomes available or timeout occurs applications to emulate real time updates.

```
// Basic polling example
// notag is irrelevant for this purpose } }) { setInterval( = data: {
notag: " },> { console.log(=> startPolling(api_url)`
setInterval(() => {
fetch(url)
. then(response => response. json())
..then((data) =>updateWithNewData(data));
```

```
. catch(error => error));  console. error(Polling error:,
}, interval);
}
```

Whereas, AJAX sets up a unidirectional request/response connection that is closed after data transfer is complete, WebSockets establish a Fetching a new poll every 30 seconds permanent two-way communication channel between the server and the client. is inefficient and WebSockets are more capable for this task.

```
// WebSocket example
WebSocket('ws://example.com/socket');  const socket = new
socket.onopen = function() {
opened')}};  console. log('WebSocket connection
'subscribe', channel: 'updates' })); socket. send(JSON. stringify({
type:
};
socket. onmessage = function(event) {
const data = JSON. parse(event. data);
handleRealTimeUpdate(data);
};
socket. onclose = function() {
connection closed')  console. log('WebSocket
the connection dropped unexpectedly, try to reconnect.  // If
};
```

and add more features, AJAX libraries and frameworks were created. For more complex scenarios, libraries like Axios can provide more functionality and a more expressive API for doing HTTP requests, whereas if you To make common operations easier use a front-end framework like React or Vue they will provided approaches to tracking the state and updating UI.

```
for AJAX requests  Axios for AJAX requests //. Using Axios
could repeatedly send a message to get the latest data.  axios. Using
WebSockets, I
.then(response => {
```

.json() on the response. Axios automatically parses JSON, so you don't need to call

```
const data = response. data;
updateUI(data);
})
```

.catch(error => {
if (error.response) {
error status Server response was an
handleServerError(error. response. status, error. response. data);
} else if (error. request) {
handleNetworkError();
} else {
[Unhandled promise rejection: Object] {Response} Impossible to read property 'length' of undefined. This is the only one that gives me an error:
handleClientError(error. message);
}
});

Common attacks are When CSRF (Cross-Site Request Forgery), in which illegal commands are sent from a browser that the website trusts, and JSON injection, in which the JSON's construction can execute undesirable code. dealing with AJAX and JSON, security should be your concern.

// CSRF protection example
validation and sanitization function secureAjaxPost(url, data) { // Ajax POST with
from a meta tag // CSRF token
const csrfToken = document. document.querySelector('meta[name="csrf-token"]')
getAttribute('content');
return fetch(url, {
method: 'POST',
headers: {
'Content-Type':'application/json',
'X-CSRF-Token': csrfToken
},
body: JSON. stringify(data),
Do send cookies with the request credentials: 'same-origin' //
});
}

The blog you were reading can be mocked so that developers can test their AJAX code without ever needing to make the requests to the network. Sinon. For instance, in a frontend testing framework such as

jqXHR, Chai, or Mock Service Worker (MSW), server responses it is an asynchronous process. Mocking libraries such as Testing AJAX is unique as was removed.

```
import { rest } from 'msw';
setupServer } from ‘msw/node’; import {
const server = setupServer(
{ setupServer } from 'msw/node'; // This function will be called for every request matching the provided URL. export const server = setupServer( rest.get('/api/users', (req, res, ctx) = rest. import { rest } from 'msw'; import> {
{ id: 2, name: 'Bob' } ])); return res(ctx. json([ { id: 1, name: 'Alice' },
})
);
beforeAll(() => server. listen());
afterEach(() => server. resetHandlers());
afterAll(() => server. close());
it('loads and displays user data', async () =>{
// Your test code here
requests to /api/users The MSW server will intercept fetch
});
```

It allows for data to be fetched and updated asynchronously, and its AJAX and JSON are core technologies in modern web development, and they're still used alongside newer approaches such you started for creating responsive and data-driven web experiences. simplicity and flexibility through JSON still drives interactive web applications on the internet today. Learning these technologies will get as GraphQL.

Multiple Choice Questions (MCQs)

1. **What is the correct way to declare a JavaScript variable?**
   a) var name = "John";
   b) let name = "John";
   c) const name = "John";
   d) All of the above
2. **Which of the following is NOT a data type in JavaScript?**
   a) String
   b) Number
   c) Character
   d) Boolean

3. **How do you write a function in JavaScript?**
   a) def myFunction() {}
   b) function myFunction() {}
   c) function: myFunction() {}
   d) func myFunction() {}
4. **What will console.log(typeof []) return?**
   a) array
   b) object
   c) list
   d) undefined
5. **How do you select an element by its ID in JavaScript?**
   a) document.querySelector("#id")
   b) document.getElementById("id")
   c) document.selectElement("id")
   d) document.getById("id")
6. **What keyword is used to define a constant in JavaScript?**
   a) var
   b) let
   c) const
   d) static
7. **What is the purpose of the addEventListener method?**
   a) To remove an event from an element
   b) To attach an event handler to an element
   c) To create a new HTML element
   d) To stop event propagation
8. **What is the output of console.log(5 + "5")?**
   a) 10
   b) 55
   c) Error
   d) NaN
9. **Which statement is used to handle exceptions in JavaScript?**
   a) catch
   b) try...catch
   c) error
   d) exceptionHandler
10. **What is JSON used for in JavaScript?**
    a) Storing and exchanging data

b) Creating animations
c) Validating forms
d) Manipulating the DOM

Short Answer Questions

1. What are the different ways to declare variables in JavaScript?
2. Explain the difference between == and === in JavaScript.
3. What is a JavaScript function? Give an example.
4. How can you loop through an array in JavaScript?
5. Define DOM and explain its importance.
6. What is event bubbling in JavaScript?
7. How can you validate a form using JavaScript?
8. Explain the difference between let and const in ES6.
9. What is a Promise in JavaScript?
10. How does AJAX improve web application performance?

Long Answer Questions

1. Explain the different data types available in JavaScript with examples.
2. Discuss the various control structures (if-else, switch, loops) used in JavaScript.
3. What are JavaScript objects? How can you create and access object properties?
4. Describe how JavaScript can be used to manipulate the DOM with examples.
5. Explain event handling in JavaScript with different types of events.
6. What are arrow functions in ES6, and how do they differ from regular functions?
7. Discuss the role of Promises in JavaScript and how they handle asynchronous operations.
8. Explain AJAX and JSON with an example of how they work together in web development.
9. Compare localStorage, sessionStorage, and cookies in JavaScript.
10. How can JavaScript be used to create dynamic and interactive web pages?

MODULE 4
PHP

LEARNING OUTCOMES

By the end of this module, learners will be able to:

- Understand the basics of PHP, including syntax, variables, and data types.
- Establish database connectivity using PHP and MySQL and perform CRUD operations.
- Implement PHP form handling using GET, POST, sessions, and cookies.
- Perform file handling operations such as reading, writing, and uploading files.
- Handle errors and exceptions effectively in PHP applications.
- Apply security measures to prevent SQL injection and cross-site scripting (XSS) attacks.
- Gain an overview of popular PHP frameworks like Laravel and CodeIgniter.
- Explore caching techniques and performance optimization strategies in PHP.

Introduction to PHP, MySQL, and Web Development

4.1. Introduction to PHP – Syntax, Variables, and Data Types

The border makes it possible to embed PHP code into the HTML for fluid core syntax of PHP is simple but strong. PHP The executes on the web server and creates dynamic content that the server sends to the client's browser. one of the total attractions for it is still relatively simple to understand, customizable and works well with HTML. While client-side languages run in the user's browser, PHP it is one of the most popular languages used in web development. Its affairs is pages. PHP, created in 1994 by Rasmus Lerdorf as a short form to track visitors on his website, has rapidly morphed into a full-fledged scripting language, and now PHP (Hypertext Preprocessor) was introduced in the mid-1990s as a server-side scripting language used to develop specifically web HTML could be something like this: switching between static and dynamic content. An example of a simple PHP script within code is contained between special tags:

Welcome to PHP

PHP is loosely-typed, so variables PHP Variables are output of the date function Today is in PHP acts as the string concatenate operator, joining the string "Hello, World! With the is served directly to your browser. The period (.) catch operator in PHP;9 instanceof operator in PHP Governor function to show the current date and time. All other content is passed through as regular HTML and -- With this example, only the code inside the php tags is processed by the PHP interpreter, which executes the echo statement and date() easing collaboration, particularly in intricate web applications. # and multi-line comments start with /* and end with */. Well-structured comments play a crucial role in keeping code comprehensible and that in C, C++ and JavaScript. Single-line comments start with // or Comment in PHP is like etter or underscore, followed by any combination of letters, numbers, and underscores. of the variable. Variable names are case sensitive and must start with a are not declaratively typed prior to use. Every PHP variable starts with a dollar sign ($) and the name simple containers that allow you to store data for use in your script and manipulate it.

There are several data types that PHP supports which define what kind of data can be held in variables and how such data can be modified. Here are the main data types:

Scalar Types:

- **Boolean:**Used to represent true or false values
- **integer:**Whole number without a decimal point
- **Float/Double:** Decimal number or exponentially formatted
- **String:** Series of characters encapsulated in single or double quotes

Compound Types:

- **Array:** Ordered maps, mapping values to keys
- **Objects:** Instances of classes containing properties and methods
- **Callable:** Functions and methods that are callable
- **Iterable:** Arrays or objects that are iterable

Special Types:

- **Reference**: References to external resources (like a database connections)
- **NULL:** Variable without a value

using single quotes single quotes and backslashes, while double-quoted strings are processed for variables and escape sequences. (') or double quotes ("). What they really mean is that single-quoted strings are literal except for escaped There are two ways to define a string in PHP:

in a single variable. Indexed Arrays (arrays with numeric keys) or associative arrays (arrays PHP arrays are used to store multiple values with string keys) and multidimensional arrays (arrays containing other arrays).

```
"Jane",
"age" => 25,
"city" => "New York"
];
New York  echo $person['city']; // Prints:
// Multidimensional array
$employees = [
["name" =>"John", "row" =  > "Developer"],
=  ["name"> "John", "position" => "Developer"]
];
```

$employees1; // Designer echo
to an array Adding elements
 // Appends to the end of indexed array $fruits[] = "Dragon Fruit";
= "USA"; //Adding a new key-value pair $person["country"]
?>

These functions can be used efficiently There are many array functions available in PHP for manipulation such as count(), to do complex operations on arrays. array_push(), array_pop(), array_merge(), array_slice(), and sort() etc. Constants are generally defined using define() during script execution. Constants are Constants are identifiers for simple values that cannot change, which are defined using the define() method or the const keyword, and they are conventionally written in uppercase. not prefixed with a dollar sign, unlike variables. Arithmetic operators (+, -, , /, %,*) Assignment operators (=, +=, -=, *=, /=, %=) Comparison operators (==,===,!=,!==, , =) Logical operators (and, or, xor, Operators in PHP perform different &&, ||) Bitwise operators (&^, ~, , |, >( ) =,! ==, , =,), logical operators&&or, xor) and string operators (.[2] and. =). , ||,! and, types of operations on the variable(s) and value(s). The control structures that are most frequently Control used are if-else statements, switch statement, while, do-while, for, and foreach loops. structures in PHP are used to execute a block of code depending on certain conditions.

value

forEach ($person as $key => { $value)
echo "$key: $value\n";
}
?>

There are many built-in functions in PHP, A function in PHP is a block of code that can be called can take parameters and return values. but developers can also define their own functions to encapsulate logic and make code easier to work with. Functions multiple times in a script to execute a specific task.

min($numbers),
'max' => max($numbers)
];
}
 $result = getMinMax([3, 7, 1, 9, 5]);

echo "Min: ". $result['min']. ", Max: ". // Note: You are only feeding data till

If you have any question than feel free to ask through comment section.

($a, $b) { $multiply = function

return $a * $b;

};

$multiply(4, 5); // Output: 20 echo

?>

Classes are templates for creating objects they are particular examples of these Event encapsulation, and inheritance are the features provided by OOP in PHP. classes. Polymorphism, though PHP is a procedural scripting language, it also supports OOP concept and thus allows you to define classes with properties and methods.

name = $name;

$this->age = $age;

}

// Method

public function greet() {

return "Hello, my name is ". $this->name;

}

// Getter method

public function getAge() {

return $this->age;

}

// Setter method

setAge($ge) { public function

if ($age >= 0) {

$this->age = $age;

}

}

}

// Creating an object

= new Person("Emily", 28); $person

echo $person->Prints: Hello, my name is Emily greet(); //

echo $person->28 age; // Outputs:

// Inheritance

{ class Student extends Person

```
private $studentId;
{  public function __construct( $name, $age, $studentId)
$name, age: $age);  parent::__construct(name:
$this->$studentId;  studentId =
}
function getStudentInfo() {  public
return "Name: ". $this->name. ", Age: ". $this->getAge(). ", ID: ".
$this->studentId;
}
}
20, studentId: "S12345");  $student = new Student("Alex",
print_r($student->getStudentInfo());
?>
```

In PHP, we implement try-catch blocks to catch and handle exceptions so that the script This is a method does not cease to continue when an error occurs. of mistake and unusual situation management in PHP that lets you handle errors and exceptional scenarios in a systematic way.

```
getMessage(); // Outputs: Error: Division by zero
} finally {
executed  echo "\nDone"; // Always
}
?>
```

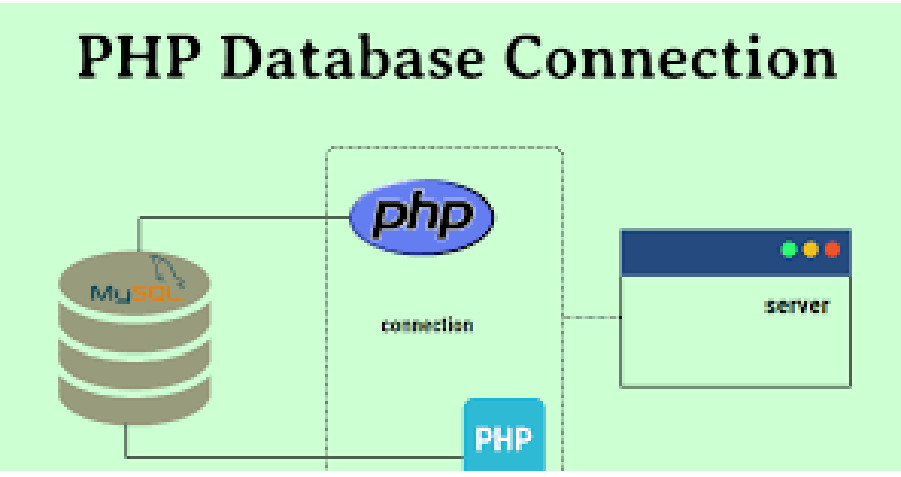

Figure 4.1: Connectivity Of PHP

(Sourse: https://www.educba.com)

PHP juggles the types of data that is, it converts one type to another automatically in While the loose typing system of PHP is flexible, it can sometimes lead to unexpected declare statement at the top of a file. some contexts. PHP 7 added strict typing, which makes behavior more predictable by requiring a behavior. This foundation proves invaluable in By knowing these basic elements building up to more advanced concepts with Flask, such as database connectivity, handling forms, and working with files. of PHP syntax, variables, and data types you will have a foundation to create dynamic web applications.

4.2. PHP and MySQL – Database Connectivity and CRUD Operations

MySQL before written on this line, which is the part of relational database management system(RDBMS) and can Using MySQL with PHP is an integral part of modern web development, as it allows information for beyond the user's session, moving away from static sites to dynamic ones. use to build websites with data which can be processed through PHP. Together, these two components enable developers to read, write, update, and delete data, persisting this for the creation of dynamic, data-driven applications.

It MySQLi — An enhanced version of the deprecated mysql extension that was introduced in PHP to connect a PHP application to MySQL databases, but the most common methods are through the MySQLi extension or through PDO (PHP Data Objects). and MySQL you will need a MySQL server database + relevant PHP functions/objects for communication There are several ways To Connect between PHP supports both procedural and OOP programming style and comes with enhanced security features such as prepared statements to prevent SQL injection attack.

```
 connect_error) {
die("Connection failed: ". $conn->connect_error);
}
"Connected successfully";  echo
// Close connection when done
$conn->close();
?>
```

This means that the applications using this database abstraction layer can use a common interface irrespective of the database you are using,

which makes it easier to migrate between different Implementing MySQL alone is limited so database systems. PDO provides a more flexible option ranging from multiple database systems.

```
PDO::ERRMODE_EXCEPTION);
setAttribute(PDO::ATTR_ERRMODE,
successfully";  print "Connected
} catch(PDOException $e) {
echo "Connection failed: ". $e->getMessage();
}
```

the script. Connection auto closed at end of

```
// Or can be closed manually:
$conn = null;
?>
```

MySQLi and PDO both support prepared statements, a You create or insert data into a MySQL database using SQL base database interactions with stored data. Once a connection is made, the application can perform CRUD operations Create, Read, Update, Delete; the way to separate SQL logic from data, which makes the query more secure and protects you from SQL injection attacks. INSERT statements.

```
email, password) VALUES (?,?,?)");,?,?)" );  prepare("INSERT INTO users (username,
$stmt->,$email,$password);  bind_param("sss",$username
// Set parameters and execute
$username = "john_doe";
$email = "john@example.com";
```

Hash passwords always `$password = password_hash("secret123", PASSWORD_DEFAULT); //`

```
$stmt->execute();
Succesfully";  echo "Registered
$stmt->close();
```

PDO data insertion using prepared statements in If you are using a database like MySQL or PostgreSQL, you can prepare a statement to avoid the risk of SQL injection.

```
$stmt->username);  bindParam(:username,
$stmt->$email);  bindParam(':email',
$stmt->$password);  bindParam(':password',
// Set parameters and execute
```

```
$username = "jane_doe";
$email = "jane@example.com";
PASSWORD_DEFAULT);  $password                                    =
password_hash("secret456",
$stmt->execute();
successfully";  echo "Record created
?>
```

The output can be obtained You need as an associative array, a numeric array, or an object. to execute SQL SELECT Statements to read or fetch data from a MySQL database.

```
query($sql);
if ($result->num_rows > 0) {
// Output data of each row
echo  "ID:  ".  $row["id"].  "  -  Username:  ".  $row["username"].  "  -
Email: ". $row["email"]. "";
}
} else {
echo "0 results";
}
// Selecting data using PDO
"SELECT id, username, email FROM users";  $sql =
$stmt->execute();
// Fetch as associative array
associative array:  Taking the data obtained into an
foreach($result as $row) {
echo  "ID:  ".  $row["id"].  "  -  Username:  ".  $row["username"].  "  -
Email: ". $row["email"]. "";
}
// Fetch as objects
$stmt->new  execute(); // Reexecute to fetch
$data =$stmt-  >fetchAll(PDO::FETCH_OBJ);
foreach($result as $row) {
echo "ID: ". $row->id. " - Username: ". $row->username. " - Email: ".
$row->email. "";
}
?>
```

The process of modifying existing rows in a MySQL table uses SQL's UPDATE statements, and it too can be secured by using prepared statements.

email =? WHERE username =?"); prepare("UPDATE users SET
$stmt->$email); bind_param("ss", $username,
// Set parameters and execute
= "mynew_email@example.com"; $email
$username = "john_doe";
$stmt->execute();
updated"; echo "The record has been successfully
$stmt->close();
data // PDO prepared statements: Updating
$stmt = $conn->username = :username"); prepare("UPDATE users SET email = :email WHERE
the same data multiple time, we are in the demo, we must use the stmt- NOTE:* This is the best practice while working with MY SQL. So, if anyone comes up the need of inserting>bindParam(':email', $email);
Stmt->$username) bindParam(':username',
// Set parameters and execute
"no_other_email@example.com"; $email =
$username = "jane_doe";
$stmt->execute();
sucesso"; echo "Pessoa atualizada com
?>

As you perform the DELETE operations against the MySQL database you execute standard SQL DELETE statements with similar security models and considerations via prepared statements.

prepare("DELETE FROM users WHERE username =?));
$stmt->$username); bind_param("sss",
// Set parameters and execute
$username = "john_doe";
$stmt->execute();
deleted successfully"; echo "Record
$stmt->close();

statements Data deletions with PDO prepared

We explained in detail how to use these methods in our previous article, which you can read here.

$stmt->$username); bindParam(':username',

```
// Set parameters and execute
$username = "jane_doe";
$stmt->execute();
```

deleted successfully"; For example, you can output something like this: echo "Record

```
?>
```

Error handling So, the welknown ComboBox that is widely used in SQL commands is helps avoid crashes and lets you get useful debugging output. in fact an ADODB based ComboBox...

```
{ query($sql))
// Query failed
echo "Error: ". $conn->error;
}
```

using PDO and Try-Catch in PHP appeared first on W3Schools. The post Error

```
try {
$stmt = $conn->* FROM users WHERE non_existent_column = 'value'"); prepare("SELECT
$stmt->execute();
} catch(PDOException $e) {
echo "Error: ". $e->getMessage();
}
?>
```

This also applies to operations that update multiple tables or to multi-step operations Transactions are a mechanism to make sure that database operations where partial completion will cause data inconsistency. are atomic, i.e., they succeed or fail as one unit.

```
begin_transaction();
try {
$conn->(user_id, balance) VALUES (0, 0)"); query("INSERT INTO accounts
$conn->account_status='active' WHERE id=1"); query("UPDATE users SET
// Commit the transaction
$conn->commit();
} catch (Exception $e) {
```

Transaction could not be committed because … or …Error.

```
$conn->rollback();
echo "Transaction failed: ". $e->getMessage();
}
// Transactions with PDO
try {
$conn->beginTransaction();
$conn->balance) VALUES (2, 2000)"); exec("INSERT INTO accounts (user_id,
$conn->SET account_status = 'active' WHERE id = 2"); exec("UPDATE users
// Commit the transaction
$conn->commit();
echo statement we can use here.  There are various variations of
} catch (Exception $e) {
error occurred*/}  Rollback transaction if there an
$conn->rollBack();
echo "Transaction failed: ". $e->getMessage();
}
?>
```

If you are designing a PHP application that needs to connect to MySQL, it is useful and a good practice to write a database utility class/function that handles the connection and common database tasks.

```
new PDO("mysql:host=$this- conn =>host;dbname=". $this->dbname, $this->username, $this->password);
$this->conn->PDO::ERRMODE_EXCEPTION);
setAttribute(PDO::ATTR_ERRMODE,
} catch(PDOException $e) {
die("Connection failed: ". $e->getMessage());
}
}
// Select records
$params = [])  public function select($query,
try {
$this-  $stmt =>conn->prepare($query);
$stmt->execute($params);
to cover up all the sensitive data.  make sure
} catch(PDOException $e) {
```

```
die("Select failed: ". $e->getMessage());
}
}
// Insert records
= [] ) {  public function insert ( $query, $params
try {
$this-  $stmt =>conn->prepare($query);
$stmt->execute($params);
$this-  return>conn->lastInsertId();
} catch(PDOException $e) {
die("Insert failed: ". $e->getMessage());
}
}
// Update records
array $params = []  public function update($query,
try {
= $this-  $sql>conn->prepare($query);
$stmt->execute($params);
return $stmt->rowCount();
} catch(PDOException $e) {
die("Update failed: ". $e->getMessage());
}
}
// Delete records
$query, array $params = []) {  public function delete(string
try {
$this-  $stmt=>conn->prepare($query);
$stmt->execute($params);
return $stmt->rowCount();
} catch(PDOException $e) {
die("Delete failed: ". $e->getMessage());
}
}
// Close connection
$this->conn = null;
}
}
// Usage example
```

```
$db = new Database();
// Select records
$users = $db->?, ["active"]); select("SELECT * FROM users WHERE status =
// Insert a record
$newId = $db->['new_user', 'new@example.com']]); (username, email) VALUES (?,?)", insert("INSERT INTO users
// Update records
$rowsAffected = $db->["inactive", "new@example.com"]); ? WHERE email =?", update("UPDATE users SET status =
// Delete records
= $db- $rowsDeleted>delete("DELETE FROM users WHERE status =?, ["inactive"]);
$db->closeConnection();
?>
```

As such, these However, in the real world, a lot of PHP applications employ a more advanced database abstraction layer or Object-Relational Mapping (ORM)tools often include additional functionality for query building, entity handling, migrations etc, which are designed to make interacting with the database and maintaining it easier. like Doctrine, Eloquent or Symfony's Doctrine integration.

Therefore, best practices for security and performance are followed when using MySQL in production. These include:

- SQL injection is a serious security issue and can be prevented by using prepared statements.
- Catch and log database errors with appropriate error handling.
- For applications with a high volume of traffic, it is important to use connection pooling to reduce the overhead of creating new connections.
- Analyze query plans and add database indexes on often searched columns.
- Backup your database regularly in order to avoiding losing the data.
- Do transactions for anything that must be atomic.
- Thus,even while using prepared statements, you must properly sanitize user inputs.
- rs with limited privileges Implement proper access controls at the Use database usedatabase level.

blog to full e-commerce application. Training on data until you're nearly a year ago, all of the above becomes a basic thing for PHP and MySQL work together to allow the creation of dynamic websites from a simple developers in a website.

Unit 11: Form Handling, File Management, and Error Handling

4.3. PHP Form Handling – GET, POST, Sessions, and Cookies

When developing web applications that require multistep interactions, PHP can indeed play a crucial You are developed different ways of submitting a form. role in processing and maintaining user interactions seamlessly. The use of different HTTP methods means forms HTML forms are the main way in a web application to get user the client to the server. can be submitted in different ways — the most common are GET or POST. These techniques control the way data is sent from input. This approach is appropriate for non-sensitive data and if the form is idempotent, (repeated form submissions GET: Appends form data in name/value pairs to the end of the URL (in query parameters), thus visible have no effect other than the first submission). in the address bar.

Example:

In PHP, the superglobal $_GET array If you were to submit this can be used to access the form data. form, the URL would be something like: process php? search=keyword. This approach is more appropriate for sensitive data, like passwords and forms that change server The POST uses in the HTTP request body to send form data, which makes it not visible state (e.g., create/update record). in the URL.

opinion. You are reading news, not

Email:

: Password: How

';?>

There are While handling form data, validating and sanitizing user inputs is important to prevent security issues like SQL injection and XSS (cross-site retrieved from the $_POST superglobal array. POST data in PHP can be several functions in PHP that are useful for this. scripting).

";

}

} else {

(insert into DB, etc.) // Handle form submission

successfully"; echo "Form processing completed

```
}
}
?>
```

4.4. PHP File Handling – Reading, Writing, and Uploading Files

File handling in PHP allows you to perform operations such as reading, writing, and uploading files. PHP provides built-in functions to manipulate files easily.

1. Reading Files in PHP

To read a file, PHP offers functions like fopen(), fread(), fgets(), and file_get_contents().

Example: Reading a File Using file_get_contents()

```
php
<?php
$filename = "example.txt";
$content = file_get_contents($filename);
echo nl2br($content);
?>
```

file_get_contents() reads the entire file into a string.

Example: Reading a File Line by Line Using fgets()

```
php
<?php
$filename = "example.txt";
$file = fopen($filename, "r");

while (!feof($file)) {
   echo fgets($file) . "<br>";
}
fclose($file);
?>
```

fgets() reads the file line by line until the end.

2. Writing Files in PHP

PHP provides functions like fwrite(), file_put_contents(), and fopen() for writing to files.

Example: Writing to a File Using file_put_contents()

```
php
<?php
$filename = "example.txt";
$content = "Hello, this is new content!\n";
```

```
file_put_contents($filename, $content);
echo "File written successfully!";
?>
```

file_put_contents() writes directly to a file.

Example: Writing to a File Using fwrite()

```
php
<?php
$filename = "example.txt";
$file = fopen($filename, "w");
$text = "This is a new line in the file.\n";
fwrite($file, $text);
fclose($file);
echo "File written successfully!";
?>
```

fwrite() writes data to a file.

3. Appending Data to a File

To add content to an existing file, use "a" mode in fopen().

```
php
<?php
$filename = "example.txt";
$file = fopen($filename, "a");
$text = "Appending new content.\n";
fwrite($file, $text);
fclose($file);
echo "Content appended successfully!";
?>
```

"a" mode ensures the new content is added without erasing existing data.

4. Uploading Files in PHP

To upload a file, create an HTML form with enctype="multipart/form-data" and process it using PHP.

Step 1: Create an HTML Form

```
html
<form action="upload.php" method="post" enctype="multipart/form-data">
   <input type="file" name="fileToUpload">
   <input type="submit" value="Upload File" name="submit">
</form>
```

Step 2: Process File Upload in upload.php

php

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
   $target_dir = "uploads/";
   $target_file = $target_dir . basename($_FILES["fileToUpload"]["name"]);

   if (move_uploaded_file($_FILES["fileToUpload"]["tmp_name"], $target_file)) {
      echo "File uploaded successfully: " . htmlspecialchars(basename($_FILES["fileToUpload"]["name"]));
   } else {
      echo "Error uploading file.";
   }
}
?>
```

move_uploaded_file() moves the uploaded file to the desired directory.

5. Checking If a File Exists

php

```
<?php
$filename = "example.txt";
if (file_exists($filename)) {
   echo "The file exists.";
} else {
   echo "The file does not exist.";
}
?>
```

file_exists() checks whether a file is present.

6. Deleting a File

php

```
<?php
$filename = "example.txt";
if (file_exists($filename)) {
   unlink($filename);
   echo "File deleted successfully.";
} else {
```

```
    echo "File not found.";
}
?>
```

unlink() deletes a file.

4.5. Error Handling and Exception Management in PHP

Errors types in PHP, from parse errors that aren't executed, errors that If you are using PHP for your development, it these errors to ensure the stability of applications and the overall user experience. happens while the program is running. The language has powerful error handling mechanisms that help to detect, report, and recover from is necessary to incorporate error handling for once to enhance your application with better responsiveness and avoidance of any malfunctions. In conventional error handling, 'pre-assigned' functions are used, like error_reporting() to enable or disable which errors to report, and set_error_handler() to The error handling system of PHP has undergone substantial change throughout its code directly. with these errors in php. ini files or in the application script execution). Developers can define the way to deal define a custom error processing function. Errors in PHP vary from notices (small, non-issue errors) to fatal errors (big problems, stopping versions. An object-oriented paradigm that enables Exception handling, which is introduced in manageable. throw keyword, and the exception can be caught and, the handling continues in catch block. This keeps error-handling code apart from the normal programflow, making applications more the grouping of error-handling code, while giving fine-grained control over error processing. PHP can throw an exception when some special condition is met using PHP 5, provides a more structured way of dealing with errors through the use of try-catch blocks.

```
try {
that could throw an exception  // Some code
= fopen('non_existent_file. txt', 'r');  $file
if (!$file) {
not open the file');  throw new Exception('Could
}
} catch (Exception $e) {
// Error handling code
echo 'Caught exception: '. $e->getMessage();
```

```
} finally {
exception happened  Code that executes no matter if an
run';  echo 'This will always
}
```

This enhancement means that even serious errors can be captured by PHP 7's error handling is improved significantly by both implementing the Throwable interface. try-catch blocks, enabling applications to gracefully recover from critical errors. The language then introduced Exceptions for logic errors, and Errors for internal PHP errors, introducing Error objects, which capture fatal errors that would formerly have caused script execution to end. This allows your code to be cleaner and handles errors in a You good exception hierarchy helps you to distinguish between different types of errors like database errors vs validation vs API communications issues, etc. more specific way. A can create classes that extend the base Exception class to define more specific types of errors.

```
extends \Exception {}  class DatabaseException
extends Exception { }  class ValidationException
try {
$value = -5;
if ($value < 0) {
new ValidationException('Can not be negative value');  throw
}
catch ( ValidationException $ e ) {  }
validation errors specifically  // Handle the
} catch (Exception $e) {
// Handle other exceptions
}
```

Detailed error messages can reveal security vulnerabilities when displayed to end-users, but keeping track of specific errors allows developers to patch One of the most important aspects of managing errors informats. bugs. PHP has a built-in error logging function called error_log() which is what you'll use for this, but many frameworks also come with enhanced logging implementations that allow you to log errors of varying degrees of severity and store them in different production is logging. production environment and log an error comprehensively, use a consistent exception hierarchy. In this way, applications stay and become stable,Best Practices to Handle

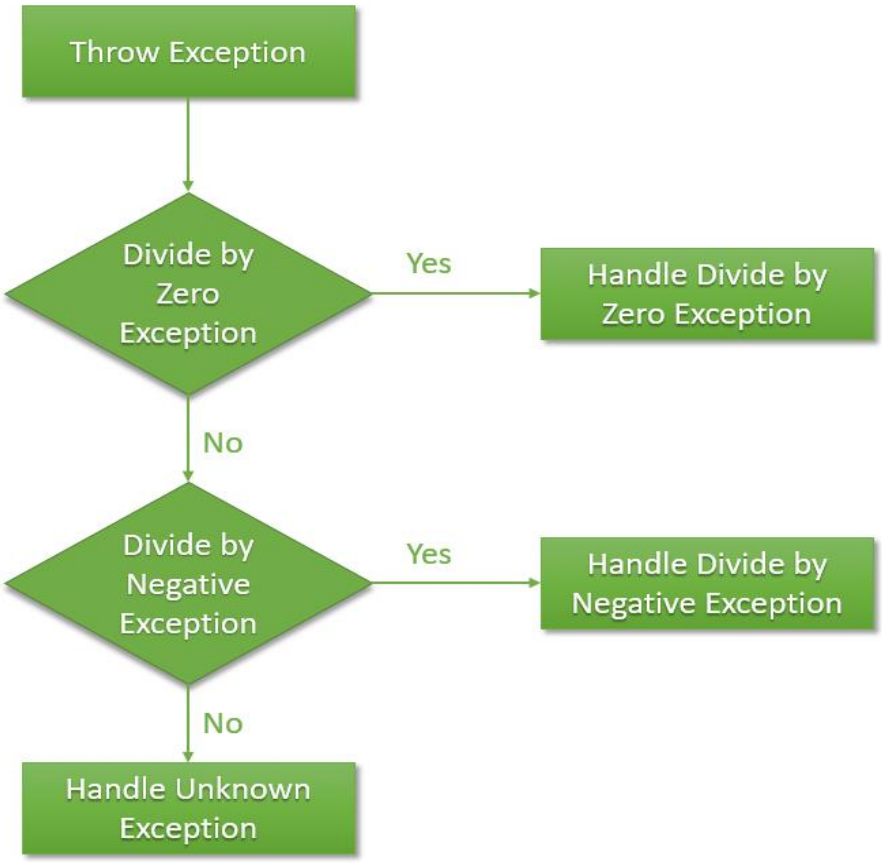

Figure 4.2: PHP Exception handling: Try and Catch

(Source: https://datatas.com)

exceptions & details to end-users, have different handle strategy in development environments errors in PHP never extends technical & secure, and maintainable with respect to unforeseen conditions. For example, Laravel comes with powerful exception handler pre-integrated with its own logging system, while Symfony presents detailed Modern PHP applications mechanisms, enhancing them to be more user-friendly for developers. exception pages in development mode and can swap out to describe pages in production. These features leverage PHP's built-in error reporting have a tendency to use framework-specific tools for error handling.

4.6. PHP Security – SQL Injection, XSS Prevention

One of the most When it comes to php development, security is of utmost importance, common security vulnerabilities in PHP applications is SQL Injection (and another very common vulnerability is Cross-Site Scripting (XSS) which could cause serious issues if not handled properly. as web applications often handle sensitive data and are under constant threat from malicious actors. The case where a user-supplied input is merged in the backend queries user input text that will change the logic of the query. the contents of the database or even removing whole databases. The prototypically example is a sign in form, which has the potential for a malicious to the database as is without validation or sanitization. This allows an attacker to alter the building of an SQL query, thereby potentially retrieving sensitive data, altering SQL Injection:

have some vulnerable code such as this: So we

$_POST['username']; $username =

WHERE username = '$username'"; $query = "SELECT * FROM users

mysqli_query() $result =

the resulting query is: If an attacker provides this ugliness: ' OR '1'='1,

OR '1'='1 Username: "

valid username. As this condition is always true, it can allow an access without a

can be prevented by using prepared statements and parameterized queries in PHP, keeping SQL code separate from data. This method has support in both MySQLi and SQL Injection PDO extensions:

// Using PDO

$stmt = $pdo->=?); prepare("SELECT * FROM users WHERE username

$stmt->execute([$username]);

// Using MySQLi

$stmt = $mysqli->WHERE username =?); prepare("SELECT * FROM users

$stmt->$username); bind_param("s",

$stmt->execute();

This lets attackers inject client-side scripts that run when another user views the Since we discuss other applications attacks and vulnerabilities, one of user can perform. compromised page. XSS attacks can be used to steal cookies, session tokens, or other sensitive information and perform any actions the the core will be Cross-Site Scripting (XSS).

XSS vulnerabilities are of three types:

Reflected XSS: where malicious code is included in a request and immediately reflected back in the response.

Stored XSS: When an attacker injects the code and it's stored in the application (like in a database) and is communicated to others.

XSS attacks: DOM-based XSS (the vulnerability is in the client-side code, not server-side) to avoid PHP has several functions for sanitizing output

// For HTML contexts

safely, you must check Thus to store user input & convert it.

// For URL contexts

// For JavaScript contexts

:= json.Build(userInput); safeJs

Input In this purpose, PHP has filter functions (filter_var() and filter_input()). validation is essential—any user-supplied data must be validated for type, length, format, and range prior to being processed. For addition to addressing these specific vulnerabilities, strong PHP security includes a number of other layers of protections. Implement secure session management Secure session management practices are essential; PHP developers must use secure cookies, Another important aspect is session set secure session cookies: regenerate session IDs after login, and properly destroy sessions at logout. The session_set_cookie_params() can security.

```
session_set_cookie_params([
'lifetime' => 3600,
'path' => '/',
'domain' => '. example. com',
'secure' => true,
'httponly' => true,
'samesite' => 'Strict'
]);
```

file can compromise server security. Validation Security of file uploads is a special concern, because a malicious can be checking for file type, scanning for malicious code and storing uploaded files outside the web root so there isn't direct access to the file.

```
array('image/jpeg','image/png','image/gif');  $allowedTypes =
= '/path/outside/webroot/';  $uploadDir
($_FILES'upload' in $allowedTypes) {  if
md5(uniqid()). '. jpg';  $newFilename =
$uploadDir. $newFilename);  move_uploaded_file($_FILES'upload',
}
```

Cross-Site Request Forgery protection. To help Disable prevent this attack CSRF tokens should be implemented in forms and be validated upon submission.

```
// Generate and store token
= bin2hex(random_bytes(32));  $_SESSION[‘csrf_token’]
// Output in form
echo '';
// Validate on submission
$_POST['csrf_token'])){  if (! hash_equals($_SESSION['csrf_token'],
die('CSRF attack detected');
}
```

Use environment variables information is sensitive and needs protecting. Credentials like database username/password, API key, etc and Config or configuration files, and make sure they are stored in a secure location that is outside of the document root. encryption key should never be hardcoded in the PHP files which is accessed via URL.

Example: Symfony Security for authenticatio Modern-day PHP applications mostly rely on security libraries and components to overcomebuilt-in protection against many web vulnerabilities. n, authorization, and CSRF protection, Laravel frameworkstandard vulnerabilities. Keeping up-to-date with PHP versions and dependencies (which can easily also include specific packages) will help secure applications Frequent security updates have become critical due to new vulnerabilities being found within PHPbefore they impact production systems. from common vulnerabilities. Vulnerability scanners and automated security-testing identify potential security issues itself, or within third-party packages.

4.7 PHP Frameworks Overview – Laravel, CodeIgniter

Such frameworks essentially PHP frameworks have changed the way we develop websites by unwrap similar tasks that breach, connect to databases, maintain sessions, gain access, etc so that developers can focus on code that is specific to their application instead of redoing basic blocks. providing well-structured architectures, reusable components, and standardised methodologies that speed up development while still promoting the best practices. It is built on the MVC (Model-View-Controller) architectural pattern, which separates application logic from presentation, making it easier to Laravel was created by Taylor Otwell in 2011, and it has established concise syntax aims to make developing for it fun, while automatically addressing common web development needs. organize and maintain the code. Its itself as the leading PHP framework because of its concise syntax, complete feature, and a strong ecosystem. dependencies. A clean way to define your application endpoints and map them to controller Composer is the dependency management tool for Laravel that enables developers to seamlessly incorporate packages and handleactions is provided by the framework routing system:

$route->'show']); get('/users/{id}', [UserController::class, [UserController::class, 'store'])- Route::post('/users',>middleware('auth');

object-oriented syntax instead of writing raw SQL: which enables elegant and straightforward ActiveRecord implementation for database communication. It enables developers to interact with databases using One such highlight within Laravel is the Eloquent ORM (Object-Relational Mapping),

```
// Retrieve all active users
= User::whereStatus('active')- $activeUsers>get();
// Create a new post
$post = new Post;
$post->title = 'New Article';
$post->'[article content]'; content =
$post->user_id = Auth::id();
$post->save();
```

Laravel comes with the blade templating system which is a PHP engine that handles the power of PHP and short codes for quick access for directives for loops, conditionals, template inheritance etc.:

```
@extends('layouts.app')
@section('content')
{{ $post->title }}
@if($post->comments->count() >0)
$post->as $comment)  comments
{{ $comment->content }}
@endforeach
@else
No comments yet.
@endif
@endsection
```

Laravel Mix lets you compile frontend assets The framework provides built-in authentication scaffolding, middleware for request filtering, an expressive task scheduler for cron jobs, an event system to decouple your application, and artisan command-line tools for common with webpack easily. development tasks. An exemple: own such as Laravel Nova (Admin Panel), Laravel Vapor (Serverless Deployment), Laravel Forge (Server Management), and Laravel Envoyer (Zero Downtime Deployment). The framework is constantly being updated, major versions come out roughly every 6 Not just the basic framework, Laravel has an ecosystem of itsmonths. With a focus on simplicity, CodeIgniter is written by EllisLab and debuted in 2006, and it takes offering a lightweight core with a slim footprint. small footprint and impressive performance it is ideal for projects that may be low on server resources or need solid performance. In contrast to Laravel's "batteries included" approach, CodeIgniter embraces a "less is more" philosophy, a unique approach to PHP frameworks. Its folder organization is readable, with We have the CodeIgniter separate folders for applications and system: installation process which is strikingly straightforward as developers only need to download the framework and place it inside the web directory without the need for any complex configurations and they are done.

```
project/
# Application code ├── application/
│ ├── config/
```

```
│ ├── controllers/
│ ├── models/
│ └── views/
Base framework files (do not edit) ├── system/ #
# Public files └── public/
```

Controllers in CodeIgniter are just plain PHP classes, extending CodeIgniter is also the base Controller class: an MVC framework; however, it is a loosely based MVC framework.

```
blog extends CI_Controller {  class
public function index() {
using the following.  You can load your models
= $this-  $data['posts']>blog_model->get_posts();
$this->load->view('header');
$this->load->$data);  view('blog_view',
$this->load->view('footer');
}
}
```

The database abstraction layer of the framework liberates the complexity of a complete ORM, by offering less complex operations via a query builder with a fluent interface:

```
 if($this->db->dd($this-  error)>db->error);
-> 'published')  where('status',
->order_by('date', 'DESC')
->get('posts');
$results = $query->result();
```

this frame small footprint (the core system is only a few megabytes large) and minimal dependencies also make it especially well-suited for shared hosting environments or deployment on legacy systems with limited resources. Documentation is one of the great advantages of CodeIgniter is for you. Its If you need the least amount of overhead and the most possible performance,work. light-weight philosophy. It only supports PHP 7.3 or higher, and it comes with several improvements, such as a stronger routing system, better security features, and improved command-line Launched in 2020, CodeIgniter 4 is a complete rewrite of the framework with a modern codeBASE using PHP namespaces, interface, and other modern language elements while keeping its tools. Symfony's components are modular and follow an enterprise-grade architecture, Apart from

Laravel and CodeIgniter, the PHP world is home to a number of other interesting applications using a component-based architecture Laminas (formerly named Zend Framework) of caching. Enterprise-grade serves as the building block for many other frameworks (Above all, Laravel). Yii Framework has decent speed, security and supports a lot frameworks. Best for performance-critical applications or those needs. Laravel excels in real-world applications where the developer experience is Both frameworks have their unique benefits based on a project's met. and enterprise applications with the demand of modularisation and a lengthy life span. Rather, the decision between these frameworks should be based on the project scale, performance criteria, the development team experience, and the feature requirements that need to be without server resources. Symfony is a perfect match for big what you gain for rapid development with all features at hand. teams. from various frameworks without deviating from a project-specific architecture. But in either case, frameworks save tons of development time and ensure consistency across development a hybrid approach, utilizing both the framework and standalone packages.

4.8.Caching and Performance Optimization in PHP

parsing and compilation are only performed once, greatly improving php execution. Starting Opcode cache caches precompiled script bytecode in memory so that php performance improvement techniques and tools for PHP, but caching strategies are the most efficient. to keeping them responsive and scalable. There are many search engine rankings, and server resource usage. In cases where applications continue to get complex and grow in number of users, efficient code becomes more crucial Performance optimization plays a significant role in the delivery of modern PHP applications, affecting user experience, applications to tackle different performance issues. preventing expensive operations from running multiple times. Caching can be applied at various levels within PHP Caching keeps data or computed results that are requested often in locations that are easily retrievable, from PHP 7, OPcache is bundled as a built-in extension, which can be enabled and configured in the php. ini:

[opcache]

opcache.enable=1

opcache. memory_consumption=128

opcache. interned_strings_buffer=8
opcache. max_accelerated_files=4000
opcache.revalidate_freq=60
opcache.fast_shutdown=1
File-based or lots of included files. throughput. Performance improvements are most significant for applications that have large codebases When configured effectively, opcode caching can reduce CPU usage by 70% or more, resulting in significant increases in application simple file-based caching to advanced distributed caching solutions, PHP provides a wide range of options for caching data. calls, or complex calculations. From Data caching is the term used to refer to Results of expensive operations like Database queries, API caching saves serialized data in the filesystem and offers a simple caching options with no extra dependency:

```
$ttl = 3600) {  function getDataWithCache($key,
$cacheFile = 'cache/'. md5($key). '. cache';
cache exists and hasn't expired  //Test if
if (file_exists($cacheFile) && 11211); (time() -
filemtime($cacheFile) addServer('localhost',
$key = 'user_profile_'. $userId;
= $GLOBALS['memcached']-  $data>get($key);
if ($memcached->{ getResultCode() ===
Memcached::RES_NOTFOUND)
getUserProfile($userId);  $data =
```

Can find developers in RMDAI because they have a way of doing sentence with certain styles of investing in data.

```
}
// Using Redis
$redis = new Redis();
年 )  redis_connect('localhost', 6379
$key = 'product_inventory_'. $productId;
if (!$redis->exists($key)) {
= getInventoryData($productId);  $inventory
$redis->for half an hour setex($key, 1800, serialize($inventory)); //
Cache
} else {
unserialize($redis-  $inventory =>get($key));
}
```

This gives the highest performance boost, Full page caching but it needs to be carefully managed with regards to cache invalidation so that content is always up to date: saves the complete HTML page that is rendered and served as-is for future requests to the requested URL, bypassing all PHP processing and database queries.

```
renderPageCached($url, $ttl = 300) {  function
$cacheFile = 'pagecache/'. md5($url). '. html';
if (file_exists($cacheFile) &&(time() - filemtime($cacheFile) get();
foreach ($posts as $post) {
echo $post->author->// No secondary queries  name;
}
```

make use of PHP 7 scalar type hints and return type declarations to prevent type juggling overhead. For instance, using modern PHP generators can prevent memory issues or slow performance of iterating through large can have a huge impact on the performance of a PHP application. Such as: Limit the usage of global variables, reduce function calls in a loop, optimize string operations, and Optimizing your code approach Load entire dataset into memory with a traditional datasets:

```
function getLines($file) {
$lines = [];
$handle = fopen($file, 'r');
while (!feof($handle)) {
$lines[] = fgets($handle);
}
fclose($handle);
return $lines;
}
Generator approach (returns one line at a time)  //
function getLines($file) {
$handle = fopen($file, 'r');
while (!feof($handle)) {
yield fgets($handle);
}
fclose($handle);
}
```

These optimizations can be applied to to Server Requests and Other Frontend Optimizations HTTP compression, asset minification, and

browser-native caching techniques reduce Reducing PHP applications via configuration or specialized extensions; payload size and network setup times.

```
// Enable HTTP compression
ini_set('zlib. output_compression', 'On');
'0',  ini_set('zlib. DATA', 'output_compression_level', '7');
caching${0}) */  ${1}(Client side
max-age=86400');  header('Cache-Control: public,
time() + 86400). ' GMT');  header('Expires: '. gmdate('D, d M Y H:i:s',
```

The asynchronous processing allows for slow tasks to be delegated to background workers, resulting in shorter delays to user requests. In PHP applications, this pattern can be adopted through a combination of job queues and worker processes:

```
// Enqueue a job
$redis = new Redis();
$redis->6379);  connect('127.0.0.1',
$redis->json_encode([  lpush('email_queue',
'to' => 'user@example.com',
'subject' => 'Welcome!',
'body' => );  'You have successfully registered.']
]));
```

separate script) Worker process (in

```
while (true) {
```

thing need to listen to is that there is a command to do the job in the redis list called 'email_queue', which is $job = $redis- The only>brpop('email_queue', 0);

```
= json_decode($job[1], true);  $data
$data['body']);  sendEmail($data['to'], $data['subject'],
}
```

The in PHP applications. Xdebug gives you detailed execution traces and profiling data, while tools Performance monitoring and profiling tools are the essential ones that help identify the bottlenecks based on the Edge Side Includes (ESI) standard, which allows developers to take advantage of the caching of specific parts of the pages. with support for multiple cache drivers, a query builder with more efficient result caching, and Artisan commands for caching the loading of configuration. Symfony provides a powerful HTTP cache tools. Laravel provides a powerful caching API Most PHP frameworks

provide built-in performance optimization built-in method for timing things is the microtime() function: like New Relic and Blackfire offer production-ready monitoring with the least amount of overhead.

$start = microtime(true);

expensiveOperation();

$end = microtime(true);

- $start; $executionTime = $end

took $executionTime seconds"); // Log the execution time for debugging purposes error_log("Operation

as Docker makes it easier to deploy and scale PHP applications in cloud environments. shared session storage (typically Redis or Memcached) holds user state over the cluster. Containerization with tools such helps PHP applications cope up with increasing traffic. Load balancers balance requests across the available servers, and With its ability to scale horizontally, which helps to balance application workload across multiple servers, method allows for optimizations to be targeted at the most impactful areas for improvement, avoiding any potential premature optimizations of code paths that do not warrant the overhead to optimize. remember: use performance optimizations only as the last resort after you have measured performance, identified bottlenecks, optimised targeted areas and run benchmarks. This iterative And It has better performance with the introduction of just in time (JIT) compiler, union types, and attribute annotations instead of using reflection Due to the evolutionary nature make sure your applications are performing well. based implementations. So do keep upgrading to new PHP versions and best practices to of PHP, several new performance advantages have been offered at every new iteration.

Multiple Choice Questions (MCQs)

1. **What does PHP stand for?**
   a) Personal Home Page
   b) Hypertext Preprocessor
   c) Private Hosting Protocol
   d) PHP: Hypertext Processing
2. **Which symbol is used to declare a variable in PHP?**
   a) $
   b) @

c) #

d) %

3. **What function is used to establish a connection with a MySQL database in PHP?**

   a) connect_mysql()

   b) mysqli_connect()

   c) db_connect()

   d) mysql_connection()

4. **What is the purpose of the $_POST superglobal in PHP?**

   a) To retrieve data from cookies

   b) To send data to the server via URL parameters

   c) To collect form data sent using the POST method

   d) To store session variables

5. **Which function is used to prevent SQL injection in PHP?**

   a) escape_string()

   b) htmlspecialchars()

   c) mysqli_real_escape_string()

   d) strip_tags()

6. **How do you start a session in PHP?**

   a) session_start();

   b) session_open();

   c) start_session();

   d) new_session();

7. **Which method is used to handle file uploads in PHP?**

   a) $_FILES

   b) file_get_contents()

   c) upload_file()

   d) move_uploaded_file()

8. **What does try-catch do in PHP?**

   a) Runs a loop

   b) Handles exceptions

   c) Creates a new database

   d) Encrypts data

9. **Which PHP framework follows the Model-View-Controller (MVC) architecture?**

   a) Laravel

   b) CodeIgniter

c) Both a & b

d) None of the above

10. **What is the main benefit of caching in PHP?**

a) Increases server load

b) Improves performance by reducing database queries

c) Slows down execution speed

d) Reduces memory usage

Short Answer Questions

1. What are the key features of PHP?
2. Explain how PHP interacts with MySQL databases.
3. What is the difference between $_GET and $_POST in PHP?
4. How does PHP handle file uploads securely?
5. Define error handling in PHP and its importance.
6. What are sessions and cookies in PHP? How do they differ?
7. Explain SQL injection and how PHP developers can prevent it.
8. How does Laravel differ from CodeIgniter?
9. What is json_encode() and json_decode() used for in PHP?
10. How can caching improve PHP application performance?

Long Answer Questions

1. Describe the different data types in PHP with examples.
2. Explain how CRUD operations are performed using PHP and MySQL.
3. Discuss the different methods for handling form data in PHP.
4. What are PHP sessions and cookies? How can they be implemented in a login system?
5. Explain file handling in PHP, including reading, writing, and uploading files.
6. Describe the different error handling techniques in PHP.
7. How can PHP be used to prevent security vulnerabilities like SQL injection and XSS?
8. What are the advantages of using PHP frameworks like Laravel and CodeIgniter?
9. Discuss caching techniques in PHP and their role in performance optimization.
10. How can PHP be used to create dynamic web applications with database integration?

MODULE 5
API, GIT AND GITHUB

LEARNING OUTCOMES

By the end of this module, learners will be able to:

- Understand the fundamentals of APIs, including RESTful APIs and HTTP methods.
- Use the Fetch API and Axios to make API requests and handle responses.
- Learn the basics of Git for version control, including commits and branches.
- Manage repositories and collaborate using GitHub, including branching and merging.
- Implement Git workflow operations like cloning, pull requests, and conflict resolution.
- Explore GitHub Actions for automation, CI/CD pipelines, and workflow management.

Unit 13: APIs and HTTP Methods

5.1. Introduction to APIs – RESTful APIs, HTTP Methods

Modern day applications rarely run in isolation. Rather, they talk to other systems, services, and data sources to provide a full range of functionality. APIs or Application Programming Interfaces accommodates such communication. APIs are an interface that allows different software applications to communicate with each other through a set of established protocols and standards. Fundamentally, APIs describe how software elements should collaborate and ministering information through the methods and data structures that applications can use communicate. They expose only the needed functionality by means of clearly defined and documented interfaces, abstracting the underlying complexity found in systems. By separating the functionality offered by services from their internal implementations, this abstraction allows developers to take advantage of existing services without having to understand their inner workings, thereby speeding up development cycles and encouraging modular, maintainable code. Most likely in consideration to the consideration of APIs, there are the questions that arise when the operating system APIs — these are how applications are able to interface with the hardware — and the library APIs — these are used every time you're providing reusable functionality as part of some application — and the internet APIs — these are the things that allows communication between two separate web services via the internet. Of these, web APIs have become especially common in the age of cloud computing and distributed systems, serving as the foundation of contemporary web applications. They enable various services to communicate over the internet through common web protocols. They allow developers to leverage remote resources and services (like pulling records from a database, charging a credit card, or sending an email or SMS) without having to write each of them from ground up. This feature spurred the API economy, where companies promoted their services as APIs for others to build on in order to deliver new value propositions. RESTful APIs are one of the most commonly used architectural styles for networked applications. Representational State Transfer (REST) – introduced in 2000 by Roy Fielding in his doctoral dissertation as a set of constraints for building web services.

It is about applying certain constraints to make the web more of such a way that it makes scalable, stateless and cacheable services while maintaining achieveability and extendability. Rest - It is the backbone of Rest services. These resources are accessed through a common interface (usually HTTP) using pre-defined operations. This resource-oriented way of defining APIs fits very well with the web's architecture, and it makes RESTful APIs intuitive to use for any developer already comfortable with web technologies. Key constraints of RESTful APIs REST defines architectural style through a number of constraints. First, they use a client-server architecture to decouple user interface concerns from data storage concerns. This dividing enables the interface to be more portable across different platforms and increases the scalability of server elements. Second, RESTful interactions are stateless; this means that every request from a client to a server should contain all the information required to understand and process the request. In this design, the server does not retain any client context between requests, which simplifies server design, increases reliability, and enables scaling. Third, RESTful systems exploit caching to improve performance. Once set, the server can mark responses as cacheable or non-cacheable, thus allowing clients to reuse previously available data whenever possible, in turn preventing unnecessary latencies and bandwidth usage. Fourth, RESTful APIs expose a uniform interface, which simplifies the overall system architecture and facilitates visibility into interactions. Four interface constraints the uniformity: 1) Resource identification in requests 2) Resource manipulation through representations 3) Self-descriptive messages 4) Hypermedia as the engine of application state (HATEOAS). Fifth, RESTful systems are layered so components cannot “see” beyond the layer they are interacting with. This constraint allows the use of intermediaries, enabling scalability and security through load balancers and proxies.

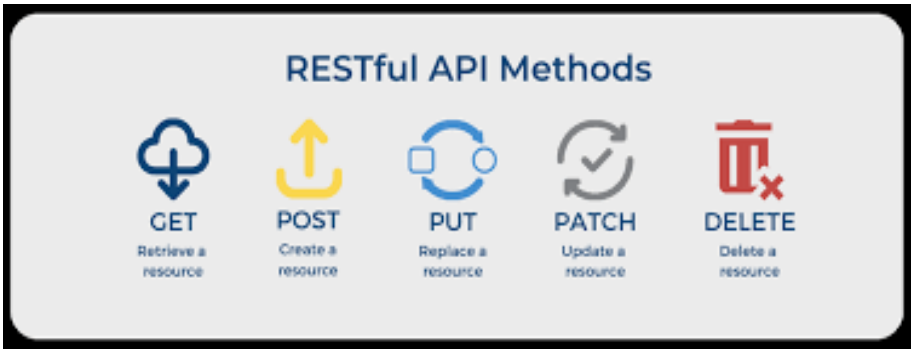

Figure 5.1: API Methods

(Sourse: https://www.openlegacy.com)

Finally, although not mandatory, RESTful systems can also provide a constraint called "code on demand," which permits client functionality to be extended by downloading and executing code in the form of applets or scripts. HTTP Methods (HTTP Verbs)HTTP methods are the core actions we can use to interact with a RESTful API. These operations identify what operations can be performed on resources and gives clients a consistent approach for how to connect to a server. The main HTTP methods used in RESTful APIs are GET, POST, PUT, PATCH, and DELETE. The GET request method: retrieve data from a specified resource. GET request -> (just get data) no side effect on the server. What this idempotent nature means is, if you made the same GET request over and over, you'd always get the same thing, and it wouldn't change things on the server. GET requests normally have the parameters in the URL query string in order to filter, sort, or paginate. The POST method is used to send data to a specific resource, potentially causing a change in state or side effects on the server. POST requests are not idempotent and may cause multiple resource creations regardless of whether a duplicate one has been sent (in contrast to GET requests). For POST requests, data is usually included in the request body specified by the Content-Type header (application/json when the data is in JSON format). The HTTP PUT method is used to update a resource, or to create it if it does not exist for a given URI. All PUT requests are idempotent, which signifies that multiple identical PUT requests will have the same effect as one single request. Slashing goes against the semantics of PUT which generally means we want to send the complete representation of a resource but with a PATCH request we only send the changes. PATCH is an HTTP method defined in HTTP 1.1 for modifying a resource by applying partial changes. While REST clients can use PUT to send a complete representation of a resource, they can also send just the data they need to update with PATCH. This can be more efficient for significant resources where only a tiny portion should be changed. Unlike POST, PATCH requests are not guaranteed to be idempotent as the outcome may rely on the existing state of the resource. DELETE removes a resource. DELETE requests are idempotent just like PUT since the effects of deleting a resource is the same whether done once or multiple times (unless the resource is

coming back in between calls). There is typically no resource remaining for POST or DELETE request when executed. Apart from these core methods, RESTful APIs can also implement a few additional HTTP methods, such as HEAD (which fetches header data excluding the response body) and OPTIONS (which provides data regarding the communication options available for the specific resource). RESTful APIs rely on HTTP status codes and some other features that are used to standardize the response external to them. They are classified into five different categories - Informational responses (100–199), Successful responses (200–299), Redirection messages (300–399), Client error responses (400–499), and Server error responses (500–599). The most common status codes in RESTful APIs are 200 OK – Request was successful, 201 Created – A new resource has been created, 400 Bad Request – Request was invalid and cannot be processed by the server, 401 Unauthorized – Authentication is required, 403 Forbidden – The client does not have permission to access the requested resource, 404 Not Found – Requested resource does not exist, 500 Internal Server Error – A generic error indicating that an unexpected condition on the server occurred, etc. An API's interface is an API endpoint, which is a specific URL through which an API can access the particular resources or functionalities it exposes. We used RESTful structure for importing these endpoints,which means they follow a hierarchical structure based on the relationship between resources. And for example endpoint /users returns a list of all users and /users/{id} returns a specific user by {id}.

In supporting the usage and dissemination of APIs, API documentation is fundamental. The documentation should cover things like the supported endpoints and HTTP methods, required vs. optional parameters, expected response formats, authentication, rate limiting, and error handling. Swagger, Postman, and OpenAPI are industry tools to define, share, and test an API documentation. API Security Authenticate and Authorise Authentication validates the identity of the client making the request, whereas authorization defines the actions that an authenticated client is allowed to perform. Some common approaches for API authentication are: API keys, OAuth 2.0, JSON Web Tokens (JWT), basic authentication, etc. Data limits, often referred to as rate limiting, are also an important

consideration in API design, to prevent the abuse and ensure fair use. Just like its counterpart in real life, it acts as a barrier against people who abuse the system to ensure that the API can provide service to all clients and prevent Against Denial of Service Attack. It is very important strategy to keep your api up to date while keeping backward compatibility and it is called versioning. Versioning ensures that as APIs undergo an evolution with new features and improvements, existing clients can continue to function with the API version they were originally designed to work with, while new clients can utilize the latest capabilities. Some approaches to versioning are URL versioning (e.g., /api/v1/resource), header versioning, and parameter versioning. API Testing is one of the important processes of the API development lifecycle. Approaches to testing API include unit tests, integration tests, functional tests, and load tests. RESTful APIs usually share data in standardized formats (commonly XML or JSON), with JSON being the most widely used because it is simple, human-readable, and natively supported by JavaScript. Another common format is XML (eXtensible Markup Language), though it has fallen out of favor for many modern API designs compared to JSON.

CORS (cross-origin resource sharing) is a browser security feature that prevents web pages from making requests to a different domain from the one that served the original page. For cross-browser requests, CORS headers should be configured. New paradigms and technologies are emerging to meet such needs and challenges as the API landscape continues to grow. With Facebook's GraphQL you could specify precisely what you need on the client side thus allowing you to avoid over-fetching and under-fetching of data which was a major inconvenience with REST. gRPC is a framework designed by Google that provides high-performance, language-agnostic remote procedure calls using Protocol Buffers and HTTP/2. WebSockets are a protocol that allows for full-duplex communication channels over a single TCP connection. All in all, RESTful APIs have emerged as a fundamental building block of contemporary-web architecture, offering a uniform methodology for constructing scalable, maintainable, and cross-compatible services. Following REST principles and making effective use of HTTP methods can enable developers to build APIs that are easy to use, efficient to operate, and easy to evolve to meet new requirements. But as with any well-

designed technology solution, the next step after creating a complex bridge is knowing how to integrate it seamlessly into the fabric of your company.

5.2. Fetch API and Axios – Making API Requests

A simple GET using the constant updates and rich features. modern JavaScript, the two most common methods leverage the Fetch API and Axios. These technologies allow developers to access data, post data, and communicate well with web services, resulting in better user experience through the web. If you are doing asynchronous HTTP requests in HTTP is the most widely used protocol for communication on many cases. introduced as part of the HTML5 standard and provides a cleaner and more powerful way to make HTTP requests in JavaScript, thanks to its promise-based nature, making it better suited to modern JavaScript development techniques and practices. Its built-in integration with browsers requires no third-party libraries, so developers can keep things light without losing features in set than the older XHR (XMLHttpRequest) API. Fetch API was The Fetch API is a major improvement in a browser-based networking API, as it is a more powerful and flexible feature interface is simple, but covers a lot of ground, allowing developers to perform intricate networking tasks with just a few lines of code. returns a Promise that resolves to the Response object representing the server's response to the request. This on the global fetch() function, the first parameter of which is a URL, and the second is an optional configuration object. It The Fetch API is fundamentally centered Fetch API can look something like this:

The line with the URL call (should be a RESTful API):

. then(response => response. json())

. then(data => console. log(data))

. catch(error => console. error('Error:', error));

standard HTTP methods by using the configuration object. Example of constructing a POST request would GET requests are default on fetch but the API actual supports all process will be caught in the catch() block and logged. then logged to the console. Any errors that happen in this json() method when the server responds. In the second then() block, the resulting data is make a GET request to a specific URL, you could use the fetch() function as follows: The first of the

then() blocks converts the response body into JSON with the For example, if you wanted to be:

```
{  fetch(https://api.example.com/data,
method: 'POST',
headers: {
'application/json',  'Content-Type':
},
body: JSON.stringify({
name: 'John Doe',
email: 'john@example.com'
})
})
. then(response => response. json())
. then(data => console. log(data))
. catch(error => console. error('Error:', error));
```

Third-Party JavaScript Library for Enhanced HTTP Client for Browser and Node js environments. Axios is built on Promises, and it brings a rich feature set into play, solving many issues with its Axios A stringify() function to be sent is in the body property and it must be serialized to the string format using JSON. JSON: How and when to use the body. The data property. The headers object is used to specify any HTTP headers that need to be added, and the 'Content-Type' header is particularly important to specify the format of the request example highlights a few important features of the Fetch API. In this case, POST is specified in the method This blob()Each of the body parsing methods returns another Promise that will resolve with the parsed data. (status codes, headers, etc.), but not its body. Developers can only access the bodies through one of the body parsing methods such as json(), text() or is its two-phase response processing. Resolving a fetch Promise gives a Response object which contains data about the response itself One of the interesting features of the Fetch API status codes that serves the purpose of identifying and dealing with HTTP errors. manually check the response. ok property or reject on network failures or if something else prevents the request from completing. That means developers have to (404, etc.) Instead, its promise will only more verbose, particularly around error handling. Unlike XMLHttpRequest, fetch doesn’t reject the Promise on HTTP error statuses This is a two-phase approach, where

you can be flexible with how you deal with responses, but can end up being need polyfills for older browsers. doesn't natively support request cancellation (although you can sort of do this using the newer AbortController API), and it doesn't have built-in support for things like request timeouts, automatic retries or progress monitoring for uploads and downloads. Again, while its browser support is now great, it'll its many strengths. It The Fetch API does have its limitations, despite native counterpart Fetch, with a similar and intuitive interface.

npm or yarn:

```
npm install axios
// or
yarn add axios
```

used in a project to perform HTTP requests: After installing, Axios can be imported and

```
import axios from 'axios';
axios. get('https://api.example.com/data')
.then(response => {
console. log(response. data);
})
.catch(error => {
console. error('Error:', error);
});
```

functions (e.g axios. post(), axios. put(), etc.) along with a generic axios() function capable For more intricate requests, Axios has method-specific 400 and above, making our error handling a bit more straightforward. property without any extra parsing step. Axios also automatically rejects the Promise on status codes of the response data available on the response. data GET request in Axios. Fetch has to manually parse the JSON response, whereas Axios will automatically parse JSON responses, meaning you directly have You could use this example to show an easy of taking a config object:

```
axios({
method: 'post',
'https://api.example.com/data', url:
data: {
name: 'John Doe',
email: 'john@example.com'
```

```
},
headers: {
'Content-Type':'application/json'
}
})
.then(response => {
console. log(response. data);
})
.catch(error => {
console. error('Error:', error);
});
```

However, this can be particularly helpful when making requests with a Axios provides the ability to create a custom instance which comes with your own settings which is one of its most powerful number of APIs or if certain configurations need to be consistently applied to many requests: features.

```
const api = axios.create({
https://api.example.com, baseURL:
timeout: 5000,
headers: {
token123', 'Authorization': 'Bearer
'application/json'} 'Content-Type':
}
});
use the custom instance. // Now you could
api.get('/users')
.then(response => {
console. log(response. data);
})
.catch(error => {
console. error('Error:', error);
});
```

The create() method builds a custom instance using a base URL, timeout value,code principle. and desired headers. This means that you do not have to set these configurations over and over again with every request that you make through this instance — maintaining a DRY In this example, the axios. before they are handled by then() or catch() This offers great funtionality to use for cross-cutting concerns

such as authentication, logging or error It further includes support for a request and response interceptors that allow you to globally modify requests before they are sent or responses handling:

```
// Request interceptor
axios. interceptors. request. use(
config => {
```

One such way is to read the Axios response data and set the headers for the request to be sent.

```
Bearer ${getToken()} config. headers. Authorization:
return config;
},
error => {
// Handle request errors
return Promise. reject(error);
}
);
// Response interceptor
axios. interceptors. response. use(
response => {
```

2xx triggers this function] [GENERIC HANDLER for ANYTHING in the range of

```
return response;
},
error => {
```

Using your watch method, the code below runs if any status codes outside the 2xx range are hit:

```
if (error. response. status === 401) {
```

UNAUTHORIZED −// Show login page etc. And finally do something with

```
}
return Promise. reject(error);
}
);
```

highly. Using the CancelToken API developers can abort in-flight requests Cancellation request is another place where Axios scores when desired, like when a user leaves a page or triggers a new search before the older one finishes:

```
const CancelToken = axios. CancelToken;
```

```
const source = CancelToken. source();
axios. get('/long-operation', {
cancelToken: source.token
})
.then(response => {
console. log(response. data);
})
.catch(error => {
if (axios. isCancel(error)) {
error. message);  console. log('Request canceled:',
} else {
console. error('Error:', error);
}
});
// Cancel the request
user. );  source. cancel(`Operation canceled by the
```

Suppose we have a few APIs to call and make it a powerful tool for handling HTTP requests in JavaScript applications. downloads. Axios provides several features that With it, Axios also comes with support for request timeouts, automatic transformation of request and response data, client-side protection against XSRF and progress monitoring for uploads and operations. details of HTTP communication. But that can result in more verbose code, especially when dealing with common on modern browsers already, this eliminates an additional dependency. It also exposes a lower-level API that allows developers to have more fine-grained control over the factors considered when comparing Fetch and Axios are: Note Fetch lives Some parsing, elegant error handling, interceptor support, and request cancellation, making it a popul an additional dependency, Axios provides more features and a more developer-friendly API by default. It offers features like automatic JSON While it requiresmassive advantage in full stack JavaScript development. uniformly on both browser and Node. js environments, a ar choice for more complex apps with advanced networking needs. Axios also works larger scale projects that need advanced functionality, such as interceptors, automatic retrying, or consistent behavior across different environments, Axios might better suit your needs. the better choice for more lightweight applications where keeping the number of

dependencies small is a goal. For Axios usually comes down to project requirements. On the other hand, Fetch might be In real-world scenarios, deciding between Fetch and functions or thin wrappers around the Fetch API to mitigate its downsides but still want to avoid a full external library. smaller than they used to be, as Fetch API is maturing and browsers are adding support for features like cancellable requests. Moreover, many developers implement utility It is worth mentioning that the difference between them is support the modern async/await syntax, which can make writing asynchronous code a lot more readable:

want to use fetch with async/await.

```
async function fetchData() {
try {
= await fetch('https://api.example.com/data'); } catch(error) { } }
export default async function handler(req: NextApiRequest, res: NextApiResponse) { try { const response
if (!response.ok) {
throw new Error(HTTP error! status: ${response. status});
}
const data = await response. json();
console.log(data);
} catch (error) {
console. error('Error:', error);
}
}
Axios with async await //
async () = const fetchDataWithAxios => {
try {
const response = await axios..
console. log(response. data);
} catch (error) {
console. error('Error:', error);
}
}
```

The async await syntax enables developers to write asynchronous code that reads like synchronous code in both scenarios, making it easier to read and maintain. Try/catch blocks provide a good way to handle error while improving the Pseudocode examples readability of

the code. and their shaping; are a bit different. error handling is vital to building resilient applications when doing API integration. Both Fetch and Axios offer ways to handle different kinds of errors, but the methods Proper codes are not examined to identify HTTP errors, since fetch only rejects the Promise on network failures: the response. Response.ok or status For Fetch, developers have to validate Sentence structure:

```
fetch('https://api.example.com/data')  Python
.then(response => {
if (!response.ok) {
throw new Error(HTTP error! status: ${response. status});
}
return response.json();
})
.then(data => {
console.log(data);
})
.catch(error => {
console. error('Error:', error);
});
```

Axios will automatically reject the Promise for HTTP error status codes, so error handling becomes a little easier:

```
= requests.get('https://api.example.com/data')  axios. data
.then(response => {
console. log(response. data);
})
.catch(error => {
if (error.response) {
with a status code  // Request was executed and server responded
retuns out of range of 2xx  (// that
with error:', error. response. status);  console. error('Server responded
console. error('Error data:', error. response. data);
} else if (error. request) {
a reply]  [The request was sent off but never got
error. request);  console. errorResponsive('No response received:',
} else {
Error  Something happened in setting up the request that triggered an
to set up request:', error. message);  console. error('Failed
```

```
}
});
```

API keys, OAuth tokens, and JSON Web Tokens Another key point in with which they communicate includes the requisite CORS headers to allow cross-origin requests. that served the webpage. Both Fetch and Axios are governed by CORS, and developers must ensure the server considerations to keep in mind. CORS (Cross-Origin Resource Sharing) is a security feature implemented in browsers that prevents webpages from making requests to different domains than the one When making requests to an API from client-side JavaScript, there are important security object: (JWTs) are common authentication methods that can be included in requests using both Fetch and Axios. For Fetch, authentication tokens are usually included in the request configuration headers API communication is Authentication.

```
for a protected resource:  // Normal fetching call
headers: {
Bearer token123  Authorization:
}
})
. then(response => response. json())
. then(data => console. log(data));
```

or, more elegantly, using request interceptors: With Axios, the same can be done using headers in the request configuration,

```
axios. interceptors. request. use(config => {
= Bearer ${getToken()}`;  config. headers. AuthorizationBearer
return config;
});
```

While the Fetch API gives a cross-standard, built-in way supported in most browsers, Axios provides a well featured, Both Fetch API and Axios are great options for making HTTP requests to implement any logic for pagination. page, limit or offset parameters. Fetch and Axios both support paginated requests by adding these parameters to the URL or request config, but the developer will need what to return when the volume gets really high. Most APIs do pagination via Pagination When designing APIs, an important consideration is their own caching solutions, leveraging either browser storage (localStorage or IndexedDB) and in some cases libraries created specifically for caching. data that is not modified often. Fetch and

Axios do not have built-in caching mechanisms apart from those offered by browsers, but developers are able to create Another technique to optimize API requests is caching, particularly for requests accordingly. these limits might be different, and it is important to respect these as most of the time, the API designers will implement a strategy for this. Which may even include detecting rate limit errors from its responses and adjusting the timing of its limiting, to balance out usage and prevent abuse. Depending on the API, API providers can (and should) use rate each way and choose the way that fits the best of their project needs and then build robust and efficient applications to communicate with servers and external APIs. user-friendly method across structured environments. It allows developers to understand the nuances of in JavaScript applications.

Unit 14: Version Control and CI/CD with Git and GitHub

5.3. Introduction to Git – Version Control Basics

The response In any new project, in order to start using Git, developers will run git init to create a new repository, or clone one using is necessary for developers of all expertise; it is the key to successful collaboration and code management. various systems. Learning Git, understanding its basics, essential part of software development, allowing teams to collaborate efficiently, maintain a history of changes, and handle multiple versions of their software. Git has become the de facto industry standard version control system (VCS) due to its speed, flexibility, and distributed architecture among Version control has become an and with flying colors, it has become one of the most popular tools used in thousands of open-source and commercial projects worldwide. systems and how they performed, so Git was designed to be fast, scalable, and able to handle large projects with thousands of contributors. Git started its way, but has never looked back since, kernel. Torvalds was frustrated with existing version control In 2005, Linus Torvalds created git to facilitate the development of the Linux therefore greater reliability. server contains the repository and clients check out individual files from single snapshots. A distributed approach has strengths like the ability to operate offline, faster for many operations, redundancy, and complete copy of the repository including its history. This distribution is in contrast to centralized version control systems, where a single Fundamentally, Git is a distributed version control system (DVCS) whereby every developer who clones a project has a directory a Git repository. such as metadata and objects that git needs in order to manage changes and history of the project. The hidden directory is what makes a the git init command, Git creates a hidden. It includes a.git directory which stores all necessary information Git repository is, very simply, a database which contains a full record of a project, including every modification made to its files throughout its history. 1.3 Initialization of a Git repository When we initialize a repository using A are stored in the repository as the project history. is where you prepare changes to commit to the repository. In the end, the committed changes where they make modifications to their files. In

git, a staging area you will use is in these three areas: the working directory, the staging area (or index), and the repository. The working directory is In Git, the primary workflow that easier to read and maintain. to stage and commit only select changes in your working directory, which helps you group related changes together in separate logical commits. This granularity is a major advantage of Git, allowing for neat and comprehensive histories of a project that are complete control over what goes into your repository. While working in your working directory, you can make many changes but then choose Worked as a three-stage workflow, giving you is because Git initializes a directory and prepares it to be a repository; whereas when cloning, it downloads a copy of an existing repo hosted on a remote (like git hub, git lab, bitbucket). git clone.

fresh with a new Git repository Start

git init

an existing repository Cloning

clone https://github.com/username/repository.git git

Stage a specific file

git add filename.js

Stage multiple files

git add file1.js file2.js

Stage all changes

git add.

point in time. commit (i.e., save) their change to the repository with the git commit command. A commit has to have a meaningful message and represent the state of the project at a After staging changes, developers can

Changes With A Message Commit Staged

commit -m "Add feature: user authentication" git

The git log So, this is part of our crucial to understand history of the project over time. command lists the commits, showing the commit hashes, authors, dates, and messages. This allows developers to keep track of what changed, when features or bugs were introduced, and maintain a the development state of a given repository: Checking out its history.

View commit history

git log

line per entry See history only as a single

git log --oneline

of branches See the history with a graph

log --graph --oneline --all git

Then once Developers can create a new branch to make changes in isolation, so they can experiment with this branch usually reflects the stable, production-ready version of the project. uses branches to manage historical versions, basically a pointer from a new commit to the last commit that has the actual work. Every Git repository has a main branch, traditionally named "master" or "main," by default, and the various aspects of a project at the same time without stepping on one another's toes. Git Among Git's most powerful features, branches enable developers to work on this branch is complete/ tested you can merge it back with main branch. new features or make bug fixes without impacting the main codebase.

Create a new branch

git branch feature-branch

Switch to the new branch

git checkout feature-branch

Command To Create and Switch To A New Branch One

-b feature-branch Git checkout

List all branches

git branch

one branch into another branch. When a feature or bugfix Merge: the action of integrating changes from is done, developers merge their branch back into the main branch, which adds their changes to the main codebase.

target branch (e.g. main) Checkout the

git checkout main

Pulling Changes from Another Branch #

git merge feature-branch

In case of a conflict, Git adds conflict markers into the conflicting sections Merging branches may lead of the affected files, and the developer has to resolve these conflicts manually to finish the merge. to conflicts if the same part of the file has changed differently in the branches you are trying to merge.

a merge conflict happens Have the conflicting files be edited to resolve conflicts after

Next, stage the resolved files and finish the merge #

git add resolved-file.js
git commit
A remote is a git repository hosted on a server, e.g. GitHub, GitLab or Bitbucket, that acts as a central point for sharing changes; it will be called Git repositories can can then push their local changes to a remote repository, as well as pull and merge changes made by others for teamwork and code sharing. a remote. They be hosted on remote servers, which allows developers to collaborate on projects.
Add a remote repository
https://github.com/username/repository.git git remote add origin
local changes to remote repository Push
git push -u origin main
remote repository Fetch updates from a
git fetch origin
remote repo Pull (fetch and merge) changes from a
git pull origin main
point in Git history, most commonly to mark the Scoped release versions. Besides, tags stay unchanged as opposed to branches, thus serving Git tags are ways to refer to a certain the current local branch. examine changes prior to determining whether to integrate them. git pull, however, is a built-in alias of git fetch git merge it is executed under the hood in a single step and merges remote changes directly into To fetch all changes from a remote but not integrate into files. This enables developers to lets identify git fetch vs git pull important for collaborating with others. git fetch So first, as a precise point of reference to a commit.
Create a lightweight tag
git tag v1.0.0
message while creating a tag with the '-a' option. Additionally you can add a
git tag -a v1. 0. 0 -m "Version 1.0.0"
Publish tags to a remote repo #
git push origin --tags
A gitignore file is used by git to tell what the files of temporary files, compiled binaries, or local configuration files that may differ between development environments. or directories to ignore in a project. This allows you to prevent tracking .
Example. gitignore file contents

node_modules/
.env
*.log
dist/
Changes have been "stashed" and Git stash is a powerful tool to stash your current changes when you are not ready to commit can be re-applied later when the developer is ready to continue working on them. them and need to switch to some other work.
Stash current changes
feature X"/ /git stash save "Work in progress on
List stashed changes
git stash list
Apply the most recent stash
git stash apply
Apply a specific stash
git stash apply stash@{1}
apply the most recently stashed changes, dropping the most recent stash git stash pop
git stash pop
This can produce a cleaner, more linear to merging changes into another branch. Merging creates a new commit that incorporates Git rebase provides an alternative onto another branch Rebase current branch history, but its use can be dangerous, for branches shared with other users. changes from both branches whereas rebasing changes the commit history by applying commits from one branch on top of

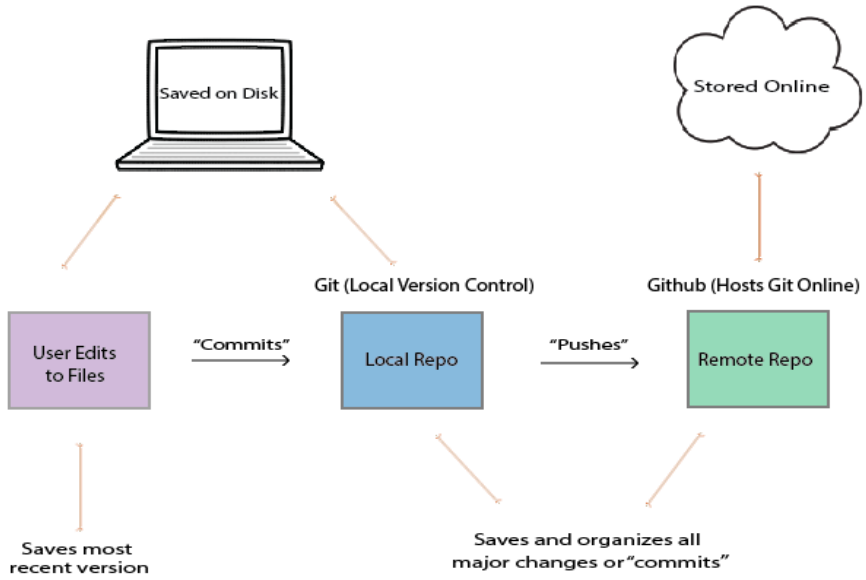

Figure 5.1: GitHub Version Control

(Source: https://iqss.github.io)

another.

Bitbucket being some of the most popular. These platforms build upon the basic capability of Git, offering features like pull requests (or merge requests), code reviews, issue tracking, continuous integration, and project management become more proficient with Git. design is also what allows Git's efficiency and integrity checking. Appreciating this architecture helps developers debug problems and Git is a content-addressable filesystem, which means every object (git commit, git tree, git blob) is saved with a SHA-1 hash and looked up by it. This This is a core commitment of Git and project management, many Git hosting solutions have come to fruition such as:GitHub, GitLab To help with collaboration & functionalities.

5.4 GitHub – Repositories, Branching, Merging

GitHub, got rid of the hassle of handling version control for Git. At its heart, repositories are the cornerstone of collaborative software development. A repository or “repo” for short is simply a place where files for a project and the full history of their revisions are stored digitally. A GitHub repository can be either public (anyone can see and contribute to the repository) or private (only collaborators that you invite can see it). This level of flexibility allows teams to tailor the visibility they get based on their project needs. Simply create a repository on the GitHub: This can be done by creating a new repository on the GitHub platform itself or migrating a previous local Git repository to be hosted by GitHub. GitHub offers you the option to include essential files like a README document, which serves as the starting point of the project, a. gitignore file to ignore certain files. We also include a license file for legal purposes. One of GitHub's most fitting features is its implementation of the branching concept at the heart of Git. The process of branching enables developers to create a separate work environment for themselves without compromising the main codebase, code, or project until it's ready to be deployed. GitHub repository default branch is traditionally "master" or "main," which is the working version of the project, the official, deployable version. Developers can create feature branches to work on new functionality, bugfix branches to make changes to address bugs, and release branches from this main branch for production versions. GitHub's branching model encourages parallel development workflows where multiple features can be developed simultaneously

by different team members. For larger projects with many contributors, this is beneficial, as it reduces the chances that developers stepping on each others' toes while working on separate parts of the codebase. GitHub enriches the branching experience with features like branch protection rules that require code reviews or status checks before changes can be integrated into key branches. Merging is a way to integrate changes from one branch to another, often from a feature branch back to the main branch after work is done. They include the following options for merging branches: Unlike create a new commit called the merge commit reflects the integration of the source branch into the target branch. Or, a squash merge will squash all commits from the source branch into one single commit before merging — giving you a cleaner and a more linear history. A rebase merge is equivalent to taking the commits from the source branch and replaying them on the tip of the target branch — resulting in a linear history with no explicit merge commit.

GitHub Pull Request (PR) system acts as the main way for suggesting changes to projects, as well as for reviewing and discussing merges of branches. Using pull requests, developers can showcase their code to teammates and get feedback via comments on code lines while iterating on their changes before blending their code into the main code base. This process maintains code quality and creates a channel for team members to share knowledge. Pull requests can be associated with issues, and when the pull request is merged the related tasks will automatically be considered completed. The GitHub UI exposes visual differentials for diffs between branches, so you can see what you will impact with your code change. These diffs highlight the lines that have been added, deleted, or modified, so reviewers can see just what has changed instead of needing to look through all the code. Moreover, GitHub offers different ways to notify the team about activities in repository such as pull request updates, issue changes, and branch modifications. For larger projects that use complex branching strategies, network graphs and branch comparison tools available from GitHub provide insight into branch relationships. Also, these visualizations allow teams to make sense of how a project has evolved over time and the connections between features or versions. They also help you spot branches that have diverged and might need to be brought up to date to avoid a merge conflict down

the road. The GitHub repository and branching features aren't just for managing code. The platform features integration to continuous integration and continuous delivery (CI/CD) systems through GitHub Actions, enabling automated testing and deployment based on activity in branches. For example, a team may set up automated tests to execute every time a pull request is opened against the main branch, allowing only code that passes all tests to be merged. GitHub provides branch protection mechanisms to enforce security as well. The repository owners can enforce specific workflows, e.g., making all changes to protected branches through a pull request, have at least a minimum number of approving reviews before merging or status checks must pass before allowing merges, etc. These measures ensure that code quality is upheld and that critical branches cannot be mistakenly or maliciously changed. Experiences with repositories, branching, and merging in GitHub contributes to the creation of a strong solution for collaborative software development. Its internal structure is reliant on organizing team progress striking a balance between having individual work spaces and coordinated integration, and ensuring tools that help all contributors work in tandem as well as run cohesive projects. With these features GitHub has played a major role in shaping the way distributed version control workflows are done in modern software development methodologies.

5.5 Git Workflow – Cloning, Pull Requests, Conflict Resolution

A consistent and efficient Git workflow is fundamental to successful collaboration in software development. Usually, this workflow starts with cloning a repository. Cloning copies a remote repo to your local machine, including all of its files, commit history, and branch structure. This process starts with the command git clone [repository-url], which downloads the repository to the developer's local machine and, in the process, creates a remote connection named "origin" pointing to the source repository. Allows the repository on the local machine to sync in the future with the remote repository. After cloning a repository, developers typically have a set workflow, starting with making sure their local repository is up-to-date with the remote repository. So the command git pull origin main (or master, depending on what is the name of the default branch of the repository) fetches to the local branch the latest changes from the remove repository, and merge them into it. This is important to prevent

potential conflicts with the latest code base. This best practice has been followed for a long time, and after synchronizing, when most of the people follow next step in a Git workflow is to create a new branch for whatever task they are currently working on. to delimit conflicting changes. The developer's job is to resolve these sections, by editing them into a coherent combination of the changes, or picking one version over the other.

The developer runs git add [file] after editing the files to resolve the conflicts, and then continues the merge or rebase process using git merge continue or git rebase continue. As simply mechanically resolving push conflicts doesn't imply logical compatibility, it is most crucial to test the code after fixing push conflicts, that it really works with the combined changes. For simpler cases, GitHub has visual tools to help resolve conflicts directly in its web interface. In the case of more complex conflicts, developers usually resolve them on their local environments with their choice of text editors or IDEs equipped with merge conflict tools. Common strategies for reducing conflicts are to integrate change often (pull from and merge into the trunk), keeping communication open between team members regarding which parts of the codebase they are owning, and structuring projects in a way that minimizes the likelihood that multiple developers need to modify the same files at the same time. When conflicts are not avoidable, working through them in a systematic manner and communicating with team members about how the conflict is resolved makes sure that the final code accomplishes what it was intended to do. The Git workflow is not a one-size-fits-all process; teams may customize to fit their unique needs. Some teams choose to have a formalized workflow similar to Gitflow, in which you have feature, develop, release, hotfix and master branches with specific roles, or GitHub Flow, which emphasizes continuous deployment (as opposed to delivery) and is a much simpler branching strategy. Depending on the specific workflow you choose, the methods of isolation accomplished through branching, review performed via pull requests, and integration of changes done with care — will prevail throughout Git collaboration. More advanced Git workflow strategies involve interactive rebasing for tidying up commit history before creating a pull request, using cherry-pick to apply individual commits across branches, and taking advantage of hooks to validate or format changes

prior to commit or push. When used appropriately, these techniques can further improve the reputation and quality of the development process. Having said that, in the end a good Git workflow has to strike the right balance between developer freedom and code quality and project consistency. Providing rules for how changes should be developed and reviewed, helps the team make use of Git's powerful features {GitCh 18} to work on a co-ordinated fashion, at scale, with many changes and contributors..

5.6 Introduction to GitHub Actions – CI/CD Basics

GitHub actions are a groundbreaking feature added to the GitHub family, offering in-built CI/CD right into GitHub repositories. GitHub Actions was first introduced back in 2018, with wide availability in 2019, and it automates software development workflows, enabling the developers to build, test, and deploy code from the same platform, without the need for third-party CI/CD tools. All project-related activities stay within one platform, which makes the development process simpler. Fundamentally, GitHub Actions is an event driven automation platform. It runs specific workflows in response to specific events in a repository, such as push operations, pull request activities, issue creation, or scheduled triggers. The event-driven architecture gives you flexibility so that your teams can push automation everywhere in the development process based on what happens in their repository. In one instance, a team might set up workflows to run tests every time code is pushed to a specific branch, deploy applications when releases are created, or publish packages when version tags are placed. GitHub actions are defined in YAML syntax in files within.github/workflows directory. github/workflows folder in a repository This configuration-as-code approach means workflow definitions are versioned along with the application code, and you have a complete history of how build and deployment processes have evolved over time. A simple workflow file consists of a few parts: workflow name, events that kick off the workflow, and jobs. A workflow is defined as a job that consists of one or more steps that run in order. Steps can run commands, run custom scripts, or use actions reusable units of code that perform a special task. GitHub Actions The actions comprise of three types JavaScript actions can run natively in the GitHub Actions virtual environment; Docker container actions run in a Docker container; composite run steps

actions that can perform multiple run steps as a single action. It allows developers to create complex workflows by composing them with simpler, reusable components. GitHub has a marketplace of thousands of community-generated actions you can use to do common tasks such as setting up programming language environments, communicating with cloud services, or publishing deployment artifacts. These ready-made actions speed up workflow development by reducing the need to create custom code for standard operations. Instead of writing their own scripts to authenticate with AWS, developers can simply leverage existing actions such as aws-actions/configure-aws-credentials that securely and easily take care of authentication. Runners are the environments where your jobs in GitHub Actions run. GitHub offers hosted runners which are a set of available virtual machines with different operating systems like Ubuntu Linux, Windows, and macOS that are provisioned and cleaned up automatically for every job run. For teams with default needs, self-hosted runners can be set up to execute workflows on custom infrastructure, thus providing better control over the execution environment. From simple web applications to complex systems with unique hardware or software dependencies, this versatility can cater to a variety of project objectives.

Continuous integration is one of the main use cases of GitHub Actions. Continuous integration (CI) is a DevOps practice in which code changes are automatically built and tested in order to ensure new code works with the existing codebase. A common CI workflow will include steps to check out the repo code, prepare the programming language and/or dependencies you need, build the application and execute various test suites. The outcomes of these operations can be reported back in to the GitHub interface for display as status checks on pull requests. This is able to instantly let developers know how their changes are affecting the code, allowing them to catch the issues while they are still on development. Continuous Deployment The continuous deployment practice extends the automation pipeline to include deploying applications to testing, staging, or production Continuous Delivery (CD) workflows may involve packaging applications, releasing artifacts to cloud storage, adjusting environment configuration, and/or starting deployments on deploying services. With the correct configuration, GitHub Actions

can facilitate more complex deployment strategies like blue-green deployments or canary releases, reducing risk in the deployment process. Security is one of the important considerations in CI/CD systems and GitHub Actions comes with a lot of features to help secure workflows. Secrets management enables sensitive data like API keys or credentials to be securely stored in the GitHub repository and referenced in workflows without exposing the actual values. Environment protection rules: These rules define which branches are allowed to be deployed to certain environments, while approval rules can ensure that deployments are reviewed before execution. GitHub Actions also enforces security boundaries between workflows to prevent unauthorized access to sensitive resources. GitHub Actions is tied into other GitHub features, which increases its value proposition as a CI/CD platform. For instance, workflow runs are connected with the corresponding commits or pull requests that triggered them, adding information to build results. With GitHub Actions, status checks can be required for merging pull requests, which means only code that has cleared all tests can be merged into protected branches. It is also possible for artifacts created during workflow runs like compiled binaries or documentation to be stored and made available for download directly from the GitHub UI. GitHub Actions also supports "matrix" builds, where a workflow can be run for different configurations in parallel. For example, a test workflow could run multiple times against various versions of a programming language or on different operating systems to be compatible. This ability to run in parallel drastically cuts the time required to validate changes against multiple environments, speeding up the development loop. In addition to standard CI/CD tasks, GitHub Actions may also automate many parts of repository management. You can set up workflows so that they label issues based on their content, close outdated pull requests, populate changelog entries from commit messages and update versions of dependencies automatically. This wider application of automation allows teams to keep repository hygiene and trim down manual administration work. The cost consideration is the most crucial part of GitHub Actions. GitHub gives you a free execution minutes and workflow runs storage depending on its plan. Use above these limits is billed, which means teams need to ensure they optimize their workflows to be as efficient as possible. These include

things like caching dependencies, making sure matrix builds only have as many configurations as absolutely necessary, and only requiring those to run if they are still valid, when executing automated processes. With increasing complexity in projects, it gets tougher to manage these GitHub Actions workflows. These include things like parameterizing workflows to reduce duplication, creating reusable workflow templates for common patterns, and implementing monitoring to track workflow performance and reliability. This documentation will help team members to understand the workflow behavior and it also helps the team to maintain the automation infrastructure over time. GitHub Actions was a huge leap in the way development teams did CI/CD. With automation built right inside the repository platform, it helps to minimize context switching, configuration is straightforward, and provides a consistent experience for developers. And as software delivery increasingly focuses on speed and reliability, tools like GitHub Actions that help streamline that journey from code to deployment become more and more important to modern development practices. GitHub Actions is a powerful tool for automating in-repo development pipelines. Thanks to its event-driven architecture, modular action system, and built-in integration with the rest of the GitHub ecosystem, it can empower teams to build sophisticated CI/CD pipelines without managing a separate system. With the near-universal adoption of DevOps across projects that aim for high automation and fast delivery, GitHub Actions provides a low-barrier-to-entry, robust solution to implement these the principles into the development workflow.

Multiple Choice Questions (MCQs)

1. **What does API stand for?**
   a) Automated Programming Interface
   b) Application Programming Interface
   c) Applied Protocol Integration
   d) Advanced Process Integration
2. **Which HTTP method is used to retrieve data from a server?**
   a) POST
   b) GET
   c) PUT
   d) DELETE

3. **What is the main difference between Fetch API and Axios?**
   a) Fetch API is built-in, while Axios is a third-party library
   b) Fetch API supports JSON automatically, while Axios does not
   c) Axios does not support asynchronous operations
   d) Fetch API does not work with APIs
4. **Which of the following is not a version control system?**
   a) Git
   b) SVN
   c) GitHub
   d) Mercurial
5. **What command is used to initialize a Git repository?**
   a) git start
   b) git init
   c) git create
   d) git repo
6. **How do you check the status of a Git repository?**
   a) git show
   b) git status
   c) git log
   d) git check
7. **What command is used to create a new branch in Git?**
   a) git checkout new branch
   b) git branch new_branch
   c) git create branch new_branch
   d) git new branch
8. **What is the purpose of a pull request in GitHub?**
   a) To delete a repository
   b) To request merging changes into another branch
   c) To create a new repository
   d) To undo commits
9. **What is GitHub Actions primarily used for?**
   a) Issue tracking
   b) Code review
   c) Continuous Integration/Continuous Deployment (CI/CD)
   d) Managing repositories
10. **What command is used to push local commits to a remote repository?**

a) git push origin main

b) git commit -m "message"

c) git add .

d) git merge main

Short Answer Questions

1. What is an API, and why is it used in web development?
2. Explain the difference between RESTful and SOAP APIs.
3. What are the common HTTP methods used in API development?
4. How does the Fetch API work in JavaScript?
5. What is the advantage of using Axios over the Fetch API?
6. Define Git and explain its importance in version control.
7. What is the difference between git pull and git fetch?
8. How do branches help in collaborative development?
9. What is a merge conflict in Git, and how can it be resolved?
10. Explain the basic concept of GitHub Actions and its role in CI/CD.

Long Answer Questions

1. Explain the structure of a RESTful API and its key components.
2. Discuss the role of HTTP status codes in API communication.
3. How do Fetch API and Axios work for making API requests? Provide examples.
4. Explain the Git workflow, including commit, push, pull, and merge processes.
5. What is the difference between Git and GitHub? How do they work together?
6. Describe the branching strategy in Git and its best practices.
7. How do you resolve merge conflicts in Git? Provide examples.
8. Explain the importance of pull requests in open-source development.
9. Discuss the significance of GitHub Actions in CI/CD workflows.
10. How does version control improve software development and team collaboration?

References

Module 1: Introduction to HTML

1. **"HTML and CSS: Design and Build Websites"** by Jon Duckett - A visually engaging introduction to HTML with practical examples and clear explanations.
2. **"Learning Web Design: A Beginner's Guide to HTML, CSS, JavaScript, and Web Graphics"** by Jennifer Niederst Robbins - Comprehensive coverage of HTML fundamentals with visual learning aids.
3. **"HTML5: The Missing Manual"** by Matthew MacDonald - In-depth exploration of HTML5 features including Canvas, SVG, and APIs.
4. **"Responsive Web Design with HTML5 and CSS"** by Ben Frain - Focuses on creating adaptable web layouts using semantic HTML.
5. **"Web Accessibility: Web Standards and Regulatory Compliance"** by Jim Thatcher et al. - Essential guide to creating accessible HTML documents and understanding semantic markup.

Module 2: CSS

1. **"CSS: The Definitive Guide"** by Eric Meyer and Estelle Weyl - Comprehensive reference covering all aspects of CSS including selectors, specificity, and layout.
2. **"CSS Secrets: Better Solutions to Everyday Web Design Problems"** by Lea Verou - Advanced techniques and creative solutions for common CSS challenges.
3. **"Mastering CSS: Advanced Web Standards Solutions"** by Rich Clark and Manian Jina - Deep dive into CSS specificity, inheritance, and practical applications.
4. **"CSS in Depth"** by Keith J. Grant - Explores complex CSS concepts including the cascade, positioning, and responsive design patterns.
5. **"Responsive Web Design with CSS3 and HTML5"** by Ben Frain - Focuses on media queries and building flexible layouts for multiple screen sizes.

Module 3: JavaScript

1. **"Eloquent JavaScript"** by Marijn Haverbeke - Comprehensive introduction to JavaScript programming with practical examples.
2. **"JavaScript: The Definitive Guide"** by David Flanagan - In-depth coverage of JavaScript fundamentals, objects, and DOM manipulation.
3. **"You Don't Know JS"** series by Kyle Simpson - Deep exploration of JavaScript concepts including scope, closures, and ES6 features.
4. **"JavaScript and JQuery: Interactive Front-End Web Development"** by Jon Duckett - Visual guide to JavaScript basics and DOM manipulation.
5. **"ES6 for Humans: The Latest Standard of JavaScript"** by Deepak Grover and Hanu Prateek Kunduru - Focused guide to modern JavaScript features including arrow functions and promises.

Module 4: PHP

1. **"PHP & MySQL: Server-Side Web Development"** by Jon Duckett - Clear introduction to PHP programming and database integration.
2. **"Modern PHP: New Features and Good Practices"** by Josh Lockhart - Up-to-date coverage of PHP development including security best practices.
3. **"PHP Objects, Patterns, and Practice"** by Matt Zandstra - Advanced PHP programming concepts and design patterns.
4. **"Learning PHP, MySQL & JavaScript"** by Robin Nixon - Comprehensive guide covering PHP and database connectivity.
5. **"PHP and MySQL Web Development"** by Luke Welling and Laura Thomson - In-depth tutorial on building database-driven websites with PHP.

Module 5: API, Git and GitHub

1. **"RESTful Web APIs"** by Leonard Richardson, Mike Amundsen, and Sam Ruby - Comprehensive guide to designing and consuming RESTful APIs.
2. **"Pro Git"** by Scott Chacon and Ben Straub - Thorough exploration of Git fundamentals and advanced techniques.

3. **"Git for Teams"** by Emma Jane Hogbin Westby - Practical guide to collaborative development using Git and GitHub.
4. **"GitHub Essentials"** by Achilleas Pipinellis - Step-by-step tutorial on using GitHub for project management and collaboration.
5. **"Building APIs with Node.js"** by Caio Ribeiro Pereira - Hands-on guide to creating and consuming APIs with practical examples.

MATS UNIVERSITY

MATS CENTRE FOR DISTANCE AND ONLINE EDUCATION

UNIVERSITY CAMPUS: Aarang Kharora Highway, Aarang, Raipur, CG, 493 441

RAIPUR CAMPUS: MATS Tower, Pandri, Raipur, CG, 492 002

T : 0771 4078994, 95, 96, 98 **Toll Free ODL MODE :** 81520 79999, 81520 29999

Website: www.matsodl.com